Universidad Carlos III Madrid

Departamento de Tecnología Electrónica

Ingeniería Industrial

Proyecto Fin de Carrera

Diseño de un Joystick USB con licencias libres.

Autor: **Félix Fabián Gómez García**

Tutor: **Luis Entrena Arrontes**

Diciembre 2011

Página dejada en blanco intencionadamente.

A mis padres,

por su paciencia infinita.

Página dejada en blanco intencionadamente.

Resumen

El documento recoge las técnicas y herramientas para crear un Joystick USB funcional a partir de un microcontrolador ATMEL USB. Todo ello haciendo uso, exclusivamente, de licencias libres.

El joystick incluye principalmente controles mediante botones, realimentación acústica y visual, así como un inclinómetro.

La memoria está planteada con carácter didáctico, pues en la medida de lo posible, se intenta que el joystick pueda ser mejorado o adaptado por otros usuarios libremente, para lo cual éstos puedan:

- Preparar un sistema operativo GNU/Linux,
- Adecuar eléctricamente los componentes a utilizar
- Escribir y dominar conceptos generales de programación de un microcontrolador
- Utilizar de los principales periféricos de un microcontrolador USB

Abstract

The present document describes the techniques and tools to create a fully functional USB Joystick from a blank USB Atmel microcontroller. All this making use, exclusively, of free licenses.

The joystick mainly includes controls by buttons, acoustic and visual feedback, as well as an accelerometer for tilt/inclination measurement.

Memory is raised with didactic nature. It is intended, as far as possible regardless of the formation of the reader, that the joystick can be freely improved or adapted by other users. To this purpose, they should be able to:

- Prepare an O.S. GNU / Linux
- Electrically adapt used components
- Write and master general concepts of microcontroller programming
- Use the main USB microcontroller peripherals

Página dejada en blanco intencionadamente.

Índice

Índice de figuras

Glosario

- **Array:** Del inglés “ristra” o matriz. Un array, por ejemplo de LEDs, es un conjunto de LEDs dispuestos para ser utilizados, conceptualmente, como un bloque.

- **Bootloader**: Del inglés “cargador de arranque”. Dispositivo físico o lógico que se encarga del arranque de un dispositivo inteligente complejo.

- **Buzzer**: Del inglés “bocina” o altavoz. Elemento electromecánico que a partir de una corriente aplicada, genera un sonido.

- **Coma flotante:** Forma de notación científica usada por los procesadores para representar números grandes y pequeños de una manera eficiente y compacta.

- **De/Multiplexar**: Combinar o “descombinar” dos o más canales en un solo medio y viceversa.

- **Debian:** Una de las principales distribuciones GNU/Linux.

- **Dispositivo inteligente:** Aquel dispositivo capaz de interpretar información o comunicaciones a partir de sus conexiones.

- **E/S:** Entrada/ Salida referida a datos. En inglés “I/O”.

- **Encoder:** Codificador rotatorio, un dispositivo electromecánico usado para convertir la posición angular de un eje a un código digital.

- **Hub:** Del inglés, concentrador. Un dispositivo capaz de intercambiar o distribuir tráfico.

- **MEMS:** Sistema Microelectromecánico. Un elemento electrónico diseñado con el fin de actuar a escala micro/nano-métrica.

- **PCB**: (Printed Circuit Board), del inglés circuito impreso. Es un medio para sostener mecánicamente y conectar eléctricamente los componentes eléctricos.

- **Perfboard:** Tipo de PCB dispuesta de agujeros a escala y distancia normalizada y, por una de sus caras, circundados por material conductor. Ideales para prototipos soldados de proyectos electrónicos.

- **Protoboard:** Placa de pruebas para elementos electrónicos reutilizable, pues consiste en una placa en la que se pueden insertar los componentes sin necesidad de soldarlos.

- **Soldadura por ola:** Es una técnica de soldadura para producción a gran escala en el que los componenetes electrónicos son soldados a la placa del circuito impreso.

- **Stack:** Del ingés, pila. En este documento se referirá a la lista ordenada de acciones que debe procesar el microcontrolador abstraída del código.

- **Teensy:** Elemento electrónico comercial, que consiste en una placa que provee de pines insertables en una protoboard a un microcontrolador Atmel USB.

Página dejada en blanco intencionadamente

1. Introducción

La evolución y crecimiento de la informática se debe, en gran parte, a la simplificación del acceso para cualquier usuario. Gracias al binomio ratón-teclado, y a los entornos de ventanas un usuario es capaz de accionar complejos sistemas informáticos abstrayéndose de toda la programación que funciona internamentes. En un entorno específico, como pueden ser aparatos médicos, sistemas de entretenimiento, sistemas de control, etc., un elemento de control diseñado para esa misión resulta fundamental.

Por otro lado, gracias al auge de internet, las licencias libres están cambiando el paradigma del conocimiento y de la técnica, pues dan lugar a una filosofía colaborativa universal y sin fricciones.

En resumen, este proyecto trata cómo desarrollar un dispositivo de control para un sistema informático, un joystick USB, utilizando únicamente licencias libres.

1.1. Motivación

Este proyecto se ha elaborado teniendo en cuenta cuatro principios fundamentales:

- Filosófico: Mostrar la potencialidad del software libre. Dar una visión de esta "novedosa" forma de creación y demostrar el estado actual de su técnica a través de un desarrollo completo de un proyecto puramente técnico.

- Didáctico: Acercar la programación software y el acondicionamiento hardware de los microcontroladores al lector. Establecer las pautas para actuar sobre los principales elementos de un microcontrolador y sobre los elementos electrónicos conectados generalmente a éstos.

- Técnico: Explicar el funcionamiento del protocolo USB e implementar una de sus aplicaciones típicas.

- Innovación: A partir de los tres principios expuestos, preparar un marco en el que desarrollar futuros dispositivos de control específicos para sistemas informáticos.

1.2. Objetivos

A lo largo del documento se detallarán las técnicas software y los medios hardware aplicados para desarrollar un dispositivo, tal que de manera autónoma, reconozca la inclinación de sus tres ejes cartesianos con respecto al vector de la gravedad, detecte pulsaciones, provea de una realimentación acústica, procese información y la envíe a un sistema informático que accionará un entorno virtual, tal como lo haría un dispositivo de interfaz humana (HID). Todo desarrollado haciendo uso exclusivo de licencias libres.

La solución propuesta constará pues de un microcontrolador, capaz de implementar el protocolo USB, que:

- Lea los valores de los tres ejes de un acelerómetro.
- Lea dos potenciómetros.
- Lea dos botones digitales.
- Accione ocho LEDs
- Genere sonidos a través de una bocina (“Buzzer”)

Mediante la conexión USB, se mostrará como un Joystick genérico (como tipo de HID seleccionado) reconocible por sistemas operativos compatibles como Windows, GNU/Linux o MacOsX y comunicará a un equipo informático estos valores.

Para demostrar su funcionamiento, el Joystick será capaz de accionar un juego, también de licencia libre, el “Secret Maryo Chronicles” (SMC)

Como resumen de lo anterior, a modo esquemático, el proyecto se podría representar tal como muestra la Figura 1. En esta figura, como se puede apreciar, se ha distinguido entre los desarrollos Software y Hardware.

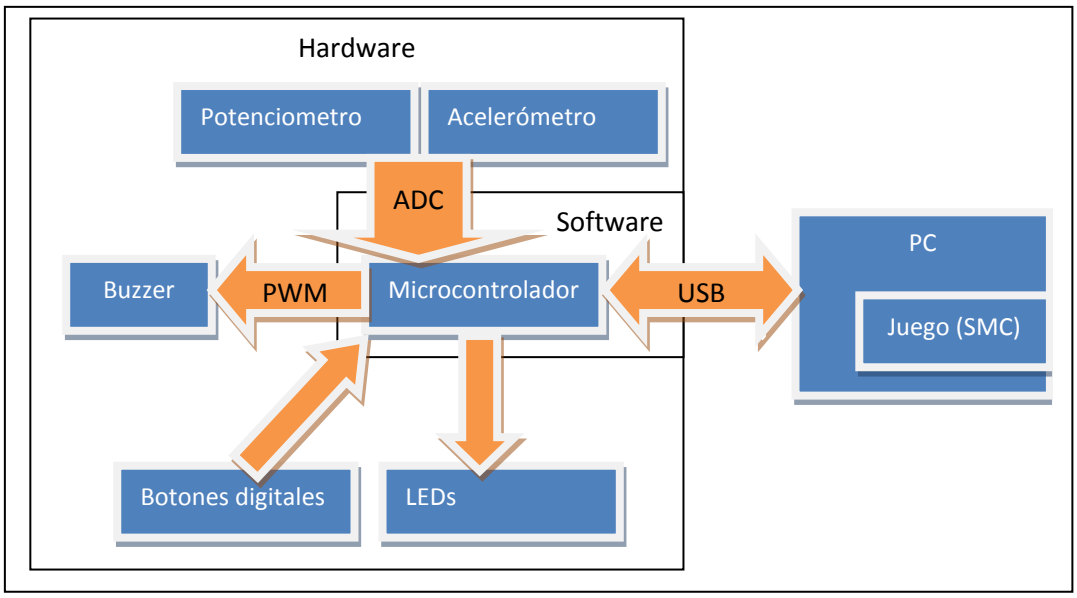

Figura 1: Resumen esquemático del proyecto

2. Estado de la técnica

2.1. HID

2.1.1. Introducción

Se entiende por "HID" o "dispositivo de interfaz humano" aquel dispositivo electrónico con el que un ser humano actúa sobre un sistema informático. Un teclado, por ejemplo, debe detectar cuando se presiona una tecla o un mando de videojuegos debe vibrar cuando el sistema informático lo considere. Se entiende, por tanto, que un HID debe procesar datos de conmutadores, botones, palancas, crucetas de dirección, rodillos, capturas de movimiento ópticas y paneles sensibles al tacto entre otras. En ocasiones, también debe poder realimentar al usuario y establecer así una comunicación bidireccional con el usuario, ya sea apagando o encendiendo alarmas visuales como en el caso de los teclados, o como se ha comentado antes, haciendo vibrar una palanca de videojuegos.

La introducción de los HID en el mundo informático empezó por la captación de señales del dispositivo a partir del puerto serie y en menor medida, del paralelo. La evolución y uso masivo de los dispositivos dio lugar al puerto PS/2 (Figura 2), un puerto basado en sus predecesores, desarrollado por IBM en 1987 específicamente para ratones y teclados. Su gran baza fue que podía ser usado junto con un modem conectado al puerto serie, pues ya no compartían controlador e interrupciones y además tenía un tamaño más reducido, muy práctico para ordenadores portátiles. No obstante presentaba ciertos problemas, entre ellos que no podía ser conectado o desconectado en caliente y que era un interfaz específico: o bien para teclado o bien para ratón, y no existió un estándar hasta que Microsoft publicó las especificaciones PC 99

Figura 2. Conexión PS/2

En la actualidad estos conectores han sido sustituidos por el USB, y se ha erigido como el interfaz predilecto para los HID por su robustez, sencillez y capacidad de expansión. También, cuando la tecnología lo permite, existe la especificación Bluetooth, que se ha convertido en un estándar *de facto* para los dispositivos inalámbricos, pero no serán motivo de estudio de este proyecto

Se puede distinguir entre HIDs comunes y menos comunes:

- **Comunes**: Teclado, ratón, trackball, touchpad y pointing stick, tablas de dibujo, joysticks, gamepads, y palancas analógicas, webcams y auriculares
- **Menos comunes**, dispositivos de simulación de conducción y de vuelo, guante cableado (Nintendo Power Glove), Surface computing, sensor de movimiento súbito (dispositivo incorporado en los Macs de Apple).

2.1.2. Adquisición de movimientos y gestos

Como se ha introducido, un HID nace de la necesidad de un ser humano de comunicarse con un sistema informático. Y, en función de la aplicación, se buscará captar distintas reacciones físicas del cuerpo. Por otro lado, la evolución de la raza humana se debe en gran parte en caminar sobre los cuartos traseros y liberar las manos para poder usar herramientas. En este sentido y en especial para este proyecto en cuestión, un HID para videojuegos, prima captar el movimiento de las manos de una persona para actuar el entorno virtual. Lo que se describe a continuación es un breve repaso de la historia de adquisición de los movimientos de las manos por parte de sistemas electromecánicos.

A principios del siglo XX, para el control de los alerones y elevadores de los aviones, se diseñó una palanca de dos grados de libertad, el "Joystick". Más adelante, este concepto fue utilizado para la captación de movimientos y gestos continuos en sistemas informáticos discretos, de tal forma que la palanca variaba la resistencia de los potenciómetros (Figura 3) y la variación de tensión de éstos era leída por un Conversor Analógico Digital (o "ADC") e interpretada, finalmente, por un microcontrolador.

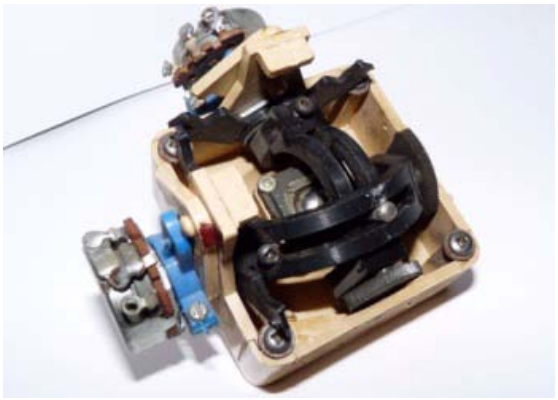

Figura 3. Ejemplo de Joystick resistivo

Con la proliferación de los sistemas de entretenimiento electro-informático el concepto de "Joystick" resistivo ha quedado obsoleto, pues la calibración y envejecimiento de éstos es muy costoso en economías de escala. En la actualidad se opta por sistemas "encoder" en cada eje a la hora de implementar un "Joystick", un sistema que comparte cierta tecnología con la captación de movimientos de los ratones ópticos.

Las nuevas tendencias apuntan un control más directo por parte del usuario, de forma que no se tenga que accionar un dispositivo, sino que el sistema informático recoja los movimientos y gestos para representarlos directamente. Es cada vez más frecuente ver pantallas táctiles de distintas tecnologías y sensores de inclinación, como soporte para todo tipo de elementos informáticos.

La adquisición de la inclinación, uno de los sistemas a estudiar por este proyecto, se consigue a través de dos tecnologías fundamentalmente: una basada en giroscopios y otra en acelerómetros. Los giróscopos, que se aplicaron inicialmente como girocompases en barcos y aviones, evolucionaron hasta "sistemas micro-electro-mecánicos" ("MEMS") tal que un material piezo-eléctrico es forzado a vibrar generando una corriente proporcional a la fuerza de Coriolis creada en el giro. Es una tecnología muy robusta, pero complicada de fabricar, y por tanto, cara.

Como nota aparte, que trasciende del alcance de este proyecto, existen pioneras formas de captación de movimiento que a partir de sistemas de "visión artificial (o por computador)" interpreta los gestos de un ser humano, como el "Kinect" de Microsoft.

2.1.3. Acelerómetros

El sistema para captación de movimientos utilizado en este proyecto es un acelerómetro dado su precio, su sencillez de integración en un circuito electrónico, su documentación y auge. A continuación se comentará la evolución y la naturaleza de estos sistemas.

Los acelerómetros tradicionalmente se han usado para la construcción, para equilibrado de maquinaria e incluso para activar el "airbag" de los automóviles. Gracias a su implementación como "MEMS" se ha logrado una reducción de costes y tamaño tal, que ahora es habitual encontrarlos en dispositivos de electrónica de consumo: sistemas anti-golpe en discos duros, reconocimiento de gestos para terminales de telefonía móvil y, por supuesto, como sensor de movimientos en sistemas de entretenimiento electrónico.

Hay varias formas de captar la aceleración mediante "MEMS", como en el caso del circuito integrado utilizado en este estudio. En él se pueden medir las diferencias de capacitancia de unos condensadores orientados ortogonalmente, en los que, por acción de la gravedad se mueve dentro de ellos una masa de prueba que hace de dieléctrico.

O también, como apunta la nueva tecnología "MEMS", mediante el acondicionamiento del traspaso térmico de la convección que se produce en una aceleración en un gas, se pueden obtener niveles de tensión proporcionales a la aceleración en cada eje. En esta tecnología, al no contar con estructuras mecánicas, se garantiza una mayor longevidad y menor envejecimiento

Existen otras formas de captar la aceleración mediante elementos electromecánicos, pero difíciles de implementar en miniatura, como son los sensores piezo-eléctricos, que albergan en su interior un elemento a presión que desarrolla una tensión proporcional a la aceleración. Y los sensores de efecto Hall, similares en concepción a los capacitivos.

Sin embargo, como se leerá a continuación, un acelerómetro no es una solución completa *per se* y necesita de una infraestructura electrónica para procesar las señales y enviarlas a un sistema que las pueda interpretar. A este sistema se le puede denominar procesado de señales de un HID.

2.2. Procesado de señales de un HID

Antiguamente los HID delegaban su control al sistema informático, es decir el HID enviaba señales al sistema informático y éste debía procesar, controlar, filtrar y realimentar esa información y traducirla a acciones a cometer por el sistema informático. Pero con la proliferación de los estándares USB, Bluetooth y el aumento de complejidad de los dispositivos y la necesidad de no sobrecargar el bus así como de cumplir estándares, conviene procesar las señales captadas antes de ser enviadas en un formato determinado.

Para entender las alternativas para procesar la información se debe explicar qué diferencias implican los dos sistemas programables por antonomasia.

a. Microprocesador

Se entiende que un microprocesador es un circuito integrado compuesto por una serie de registros, una unidad de control, una o varias unidades aritmetico-lógicas y puede tener, como es el caso de los ordenadores personales, una unidad en coma flotante. Para conseguir un sistema funcional de procesamiento de datos, el microprocesador, en un PC por ejemplo, debe conectarse a través de sus buses de procesador a los distintos elementos cuyas direcciones residen en el "chipset" de la "placa base". Estos elementos son la memoria de programa (también conocida como memoria RAM), a los dispositivos de almacenamiento (los discos duros) y a los distintos controladores de puertos de periféricos y tarjetas integradas (como en el caso de las tarjetas de video). En la figura 4, se puede apreciar la descentralización de los elementos.

Es por tanto un elemento dependiente de un sistema mayor diseñado con una filosofía modular para facilitar la especialización de los fabricantes, pero ineficiente para aplicaciones "sencillas".

Figura 4. Fotografía de un microprocesador en una placa base

b. Microcontrolador

Un microcontrolador, al contrario que el microprocesador, pretende ser autónomo. Es decir, en el mismo encapsulado se incluyen las tres funcionalidades de un ordenador, un procesador, memoria y unidades de entrada y salida. Además suelen contar, en función del modelo, distintos periféricos internos y memorias como se estudiará en siguientes apartados.

En el anexo, apartado 7.1, se puede encontrar una breve resumen de la historia, funcionamiento interno y arquitecturas.

2.2.1. Periféricos Internos de un microcontrolador

Se entiende como periféricos internos todas aquellas funcionalidades independientes de la CPU incluidas en los microcontroladores. Estas funcionalidades trabajan con registros, una posición de memoria definida, para su configuración, cargar datos de entrada y/o para cargar la salida de datos.

a. Temporizadores y contadores.

Los temporizadores y los contadores son circuitos síncronos para el conteo de pulsos. Si estos pulsos proceden del reloj interno se utilizará como temporizador. Si proceden de una fuente externa, como una variable o un PIN de E/S se utilizarán como contador.

Es un periférico casi imprescindible para la medición y la comunicación con el mundo exterior y es habitual que tenga asociada alguna interrupción. Típicamente en los microcontroladores suele haber un mínimo de dos.

b. Conversor analógico digital

Un conversor analógico digital, o "ADC", es un periférico capaz de analizar una tensión analógica en un PIN determinado a partir de una tensión interna o externa, y dar un valor digital proporcional.

Los factores diferenciadores de estos conversores, entre ellos, son la velocidad y el número de bits de precisión.

c. Comparador analógico.

Un comparador analógico compara dos valores de entrada en un amplificador operacional en lazo abierto (figura 5), uno en el terminal positivo y otro en el negativo. Cuando la tensión en el terminal positivo es mayor que en el negativo se produce una salida (V_{out}), que en el caso de un microcontrolador, modifica un registro.

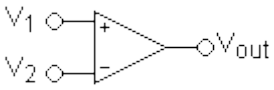

Figura 5. Representación de un amplificador operacional

Esta salida suele ser utilizada para lanzar interrupciones, bien por flanco de subida, bajada o cambio.

d. Puertos serie

Es un periférico presente en casi cualquier microcontrolador y su función común suele ser la comunicación con otro "dispositivo inteligente", bien un ordenador o un circuito integrado capaz de una comunicación serie. Este periférico, además de establecer parte de la capa física para la comunicación, se encarga de transformar la información para enviarla bit a bit. Puede establecer una comunicación bi-direccional (de entrada y salida) y en función de la tecnología puede ser síncrona, asíncrona y puede tener, o no, corrección de errores y bits de conformidad.

Algunos ejemplos comunes de estos puertos

- **UART/USART.** "Universal Asynchronous Receiver-Transmitter" o "Universal Synchronous Asynchronous Receiver-Transmitter", puerto serie bidireccional asíncrono o síncrono, respectivamente. Utilizado generalmente para la comunicación con otro microcontrolador o con un PC a través del interfaz RS-232, (erróneamente denominado puerto serie).

- **SPI.** Un puerto de comunicaciones síncrono estándar que sirve para controlar circuitos integrados que acepten un flujo de bits serie regulado por un reloj y que no sean demasiado rápidos. Es un bus que se caracteriza por minimizar el número de conductores y el tamaño del integrado.

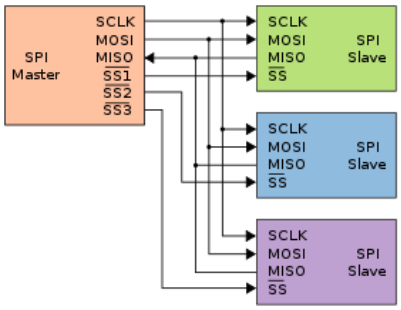

Figura 6. Esquema de comunicación SPI

El hardware consiste en: señales de reloj (SCLK), data in (MOSI), data out (MISO) y chip select (SS) para cada circuito integrado que tiene que ser controlado (figura 6). Casi cualquier dispositivo digital puede ser controlado con esta combinación de señales.
Permite una comunicación "Full Duplex", con un protocolo sencillo que se puede controlar desde el tamaño de los bloques hasta su significado. En contra tiene que no existe señal de asentimiento ("ACK"), por lo que un maestro puede estar comunicándose con un esclavo que no existe, consume un PIN del maestro por cada esclavo y sólo funciona en distancias cortas.

- **I2C (TWI).**Es un bus de comunicaciones serie síncrono de 100Kbits/segundo. Su gran peculiaridad es que utiliza tan sólo dos líneas para transmitir la información, una para datos y otra para reloj (figura 7), y además permite la existencia de varios maestros, no necesariamente fijos.

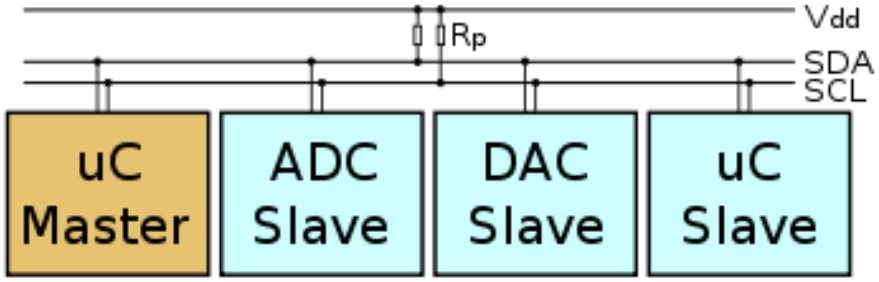

Figura 7. Esquema de conexión I2C/TWI

En contra presenta, que necesita enviar la dirección del dispositivo al que se comunica (desde 0 - 127) en su ya lento bus y que consume más potencia que por ejemplo el SPI, pues sus lineas están configuradas en drenador abierto, es decir, conectadas a la alimentación por resistencias.

Pese a no ser una patente registrada, el fabricante “Atmel” a renombrado este bus para su uso como “Two Wires Interface” (interfaz de dos cables “TWI”).

e. PWM. Modulador de ancho de pulsos

La modulación por ancho de pulso consiste en variar el ciclo de trabajo de una señal periódica (este concepto se desarrollará en el apartado 3.2.6). En el caso de los microcontroladores este periférico, apoyado por temporizadores, es capaz de variar la frecuencia, el tiempo de cresta y por extensión, el de valle, de una señal cuadrada (Figura 8).

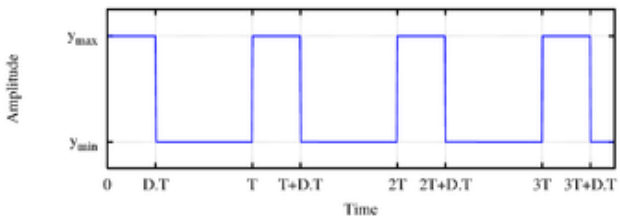

Figura 8. Señal creada por PWM

Resulta muy útil para controlar motores eléctricos, para la conversión digital a analógica y generar tonos en un zumbador o variar la potencia suministrada a un motor.

f. Bootloader, leer mientras se escribe.

El “bootloader” es una característica habitual en los microcontroladores con características para USB y, en esencia, permite que un fragmento de código permanezca e incluso programe otras partes de la memoria EEPROM con código a ejecutar. Es decir, es un periférico con su propia memoria reservada que hace las veces de interfaz para programar el microcontrolador.

Es especialmente útil, pues debidamente configurado, no se necesita más que un cable USB para programar un microcontrolador capaz de USB

g. Memoria de datos no volátil

Muchos microcontroladores han incorporado la memoria de datos no volátil como un periférico más, para el almacenamiento de datos de configuración o de los procesos que se controlan. Esta memoria es independiente de la memoria de datos tipo SRAM o la memoria de programas, en la que se almacena el código del programa a ejecutar por el procesador del microcontrolador.

Generalmente se incluye este tipo de memoria en forma de memoria EEPROM, incluso algunos de ellos permiten utilizar parte de la memoria de programa como memoria de datos no volátil, por lo que el procesador tiene la capacidad de escribir en la memoria de programas como si ésta fuese un periférico más.

Esta memoria resulta especialmente útil, ya no por su acceso distinguido del resto, sino por la capacidad de almacenar datos tomados o creados durante la ejecución del programa y que persistirán aunque se desconecte el circuito

En el siguiente apartado se estudiarán las distintas tecnologías utilizadas como memorias.

2.2.2. Memorias de un microcontrolador

Existen distintos tipos de memorias, a nivel físico, y en un microcontrolador es habitual encontrar combinaciones de ellas:

a. **ROM con máscara:** Es una memoria no volátil de sólo lectura, cuyo contenido se graba durante la fabricación del chip. El término máscara viene de la forma como se fabrican los circuitos integrados. El alto precio del diseño de la máscara sólo hace aconsejable el empleo de los microcontroladores, con este tipo de memoria, cuando se precisan cantidades superiores a varios miles de unidades.

b. **OTP:** Es una memoria no volátil de sólo lectura "programable una sola vez" por el usuario. OTP (One Time Programmable). Es el usuario quien puede escribir el programa en el chip mediante un sencillo grabador controlado por un programa desde un PC. La versión OTP es recomendable cuando es muy corto el ciclo de diseño del producto, o bien, en la construcción de prototipos y series muy pequeñas.

c. **EPROM: Erasable Programmable Read Only Memory**, pueden borrarse y grabarse muchas veces. La grabación se realiza, como en el caso de los OTP, con un grabador gobernado desde un PC. Si, posteriormente, se desea borrar el contenido, disponen de una ventana de cristal en su superficie, por la que se somete a la EPROM a rayos ultravioleta durante varios minutos. Las cápsulas son de material cerámico y son más caros que los microcontroladores con memoria OTP, que están hechos con material plástico. Hoy día se utilizan poco, siendo sustituidas por memorias EEPROM o Flash.

d. **EEPROM: Electrical Erasable Programmable Read Only Memory**, son memorias de sólo lectura, programables y borrables eléctricamente EEPROM a través de la aplicación de una tensión de predisposición Vpp. Tanto la programación como el borrado se realizan eléctricamente desde el propio grabador y bajo el control programado de un PC. Es muy cómoda y rápida la operación de grabado y la de borrado. No disponen de ventana de cristal en la superficie. Los microcontroladores dotados de memoria EEPROM una vez instalados en el circuito, pueden grabarse y borrarse cuantas veces se quiera sin ser retirados de dicho circuito. Para ello se usan "grabadores en circuito" que confieren una gran flexibilidad y rapidez a la hora de realizar modificaciones en el programa de trabajo. Esta tecnología de memoria está empezando a ser sustituida por memorias tipo Flash, o al menos, sirven como complemento

e. **SRAM, Memoria Estática de Acceso Aleatorio.** Es una tecnología de memoria capaz de de mantener los datos sin necesidad de un circuito de refresco y que por sus características, es fundamentalmente usada de manera muy específica
Son memorias volátiles, es decir, pierden la información si se interrumpe la corriente. Otra característica de estas memorias es que son de acceso aleatorio, lo que significa que las posiciones de memoria pueden ser escritas o leídas en cualquier orden. Resulta el tipo de memoria ideal para el almacenaje de variables creadas durante la ejecución del programa.

2.2.3. Posiciones de memoria

De manera simplista y licenciosa, se puede concretar que los microcontroladores actuales tienen dos tipos de memoria: la memoria del programa ubicada en una memoria de tipo EEPROM y que se ejecuta secuencialmente desde que se activa el microcontrolador, y la memoria de datos, que almacena datos de variables durante la ejecución del programa. Estas memorias tienen un direccionamiento fijo, y parte de esas direcciones tienen usos concretos:

a. Vectores de Interrupción: Ocupan una posición fija de la memoria de programa, y representan la dirección a la que se dirige la ejecución del programa cuando se produce una interrupción.
Por ejemplo, si se produce la interrupción externa “INT0”, el programa para su ejecución, almacena la posición de programa y se dirige a la posición de memoria “$0002”. Tras atender a la interrupción, recupera la dirección almacenada y retoma su marcha por donde iba.

b. Registros. Son un espacio fijo de la memoria SRAM muy reducido pero de vital importancia. En ellos se almacenan los datos de configuración de los distintos elementos del microcontrolador.
Por ejemplo, con el registro ADCSRA situado en la posición de memoria “0x7A” se puede encender, apagar, habilitar o configurar el convertidor analógico/digital interno de un microcontrolador “Atmel”.
Además se utilizan para ingresar datos que puede utilizar un periférico interno o para guardar la salida del mismo. Por ejemplo, la “unidad aritmético lógica” carga los operadores de un registros y almacena el resultado en otro.
Los registros, o más bien su tamaño, determinará en gran medida la potencia de un procesador. Pues si, por ejemplo, la “unidad aritmético lógica” es capaz de trabajar con registros de 16 bits en vez de con 8 bits, para realizar una operación de 128 bits, tardará la mitad de operaciones. Entonces se entiende que el tamaño de los registros estará acorde con todos los elementos del microcontrolador

2.2.4. Familias de microcontroladores

Existen muchas familias y fabricantes de microcontroladores, como los 68HC de Freescale, los H8 de Hitachi, MCS de Intel, los COP8 de National, ST62 de STMicroelectronics, TMS70 de Texas Instrument. Sin embargo, los más utilizados por los aficionados o estudiantes son los PIC de Microchip y los AVR de Atmel.

a. PIC

Breve historia PIC

Los PIC son una familia de microcontroladores tipo RISC fabricados porMicrochip Technology Inc. y derivados del PIC1650, originalmente desarrollado por la división de microelectrónica de General Instrument.

El PIC de 8 bits se desarrolló en 1975 para mejorar el rendimiento del sistema quitando peso de E/S a la CPU. El PIC utilizaba microcódigo simple almacenado en ROM para realizar estas tareas; y aunque el término no se usaba por aquel entonces, se trata de un diseño RISC que ejecuta una instrucción cada 4 ciclos del oscilador.

En 1985 la división de microelectrónica de General Instrument se separa como compañía independiente que es incorporada como filial (el 14 de diciembre de 1987 cambia el nombre a Microchip Technology y en 1989 es adquirida por un grupo de inversores) y el nuevo propietario canceló casi todos los desarrollos, que para esas fechas la mayoría estaban obsoletos. El PIC, sin embargo, se mejoró con EPROM para conseguir un controlador de canal programable. Hoy en día multitud de PICs vienen con varios periféricos incluidos (módulos de comunicación serie, UARTs, núcleos de control de motores, etc.) y con memoria de programa desde 512 a 32.000 palabras (una palabra corresponde a una instrucción en ensamblador, y puede ser 12, 14 o 16 bits, dependiendo de la familia específica de PICmicro).

Estructura del microcontrolador PIC

La arquitectura del PIC es sumamente minimalista. Esta caracterizada por las siguientes prestaciones:

- Área de código y de datos separadas (Arquitectura Harvard).
- Un reducido número de instrucciones de longitud fija.
- La mayoría de las instrucciones se ejecutan en un solo ciclo de ejecución (4 ciclos de reloj), con ciclos de único retraso en las bifurcaciones y saltos.
- Todas las posiciones de la RAM funcionan como registros de origen y/o de destino de operaciones matemáticas y otras funciones.
- Una pila para almacenar instrucciones de regreso de funciones.
- Una relativamente pequeña cantidad de espacio de datos direccionable (típicamente, 256 bytes), extensible a través de manipulación de bancos de memoria.

- El espacio de datos está relacionado con el CPU, puertos, y los registros de los periféricos.
- El contador de programa está también relacionado dentro del espacio de datos, y es posible escribir en él (permitiendo saltos indirectos).

A diferencian de la mayoría de otras CPUs, no hay distinción entre los espacios de memoria y los espacios de registros, ya que la RAM cumple ambas funciones, y esta es normalmente referida como "archivo de registros" o simplemente, registros.

Se pueden considerar tres grandes gamas de MCUs PIC en la actualidad: Los básicos (Linebase), los de medio rango (Mid Range) y los de altas prestaciones (high performance). Los PIC18 son considerados de alto rendimiento y tienen entre sus miembros a PICs con módulos de comunicación y protocolos avanzados (USB, Ethernet, Zigbee por ejemplo).

Recursos de información PIC

Se puede encontrar mucha información y documentación sobre PICs en Internet principalmente por dos motivos: el primero, porque han sido muy usados para romper los sistemas de seguridad de varios productos de consumo mayoritario (televisión de pago, Play Station...), lo que atrae la atención de los crackers; y segundo, porque el PIC16C84 fue uno de los primeros microcontroladores fácilmente reprogramables para aficionados. Hay muchos foros y listas de correo dedicados al PIC en los que un usuario puede proponer sus dudas y recibir respuestas.

b. **AVR**

Historia AVR

Los AVR son una familia de microcontroladores RISC de Atmel. La arquitectura de los AVR fue concebida por dos estudiantes en el Norwegian Institute of Technology, y posteriormente refinada y desarrollada en Atmel Norway, la empresa subsidiaria de Atmel, fundada por los dos arquitectos del chip.

Figura 9. Logotipo de Atmel

Estructura AVR

El AVR es una CPU de arquitectura Harvard. Tiene 32 registros de 8 bits. Algunas instrucciones sólo operan en un subconjunto de estos registros. La concatenación de los 32 registros, los registros de entrada/salida y la memoria de datos conforman un espacio de direcciones unificado, al cual se accede a través de operaciones de carga/almacenamiento. A diferencia de los microcontroladores PIC, el stack se ubica en este espacio de memoria unificado, y no está limitado a un tamaño fijo.

El AVR fue diseñado desde un comienzo para la ejecución eficiente de código C compilado. Como este lenguaje utiliza profusamente punteros para el manejo de variables en memoria, los tres últimos pares de registros internos del procesador, son usados como punteros de 16 bit al espacio de memoria externa, bajo los nombres X, Y y Z.

Esto es un compromiso que se hace en arquitecturas de ocho bit desde los tiempos de Intel 8008, ya que su tamaño de palabra nativo de 8 bit (256 localidades accedidas) es pobre para direccionar. Por otro lado, hacer que todo el banco superior de 16 registros de 8 bit tenga un comportamiento alterno como un banco de 8 registros de 16 bit, complicaría mucho el diseño, violando la premisa original de su simplicidad. Además, algunas instrucciones tales como 'suma inmediata' ('add immediate' en inglés) faltan, ya que la instrucción 'resta inmediata' ('substract immediate' en inglés), con el complemento dos, puede ser usada como alternativa.

Programación AVR

El set de instrucciones AVR está implementado físicamente y disponible en el mercado en diferentes dispositivos, que comparten el mismo núcleo AVR pero tienen distintos periféricos y cantidades de RAM y ROM.

Los microcontroladores AVR tienen una cañería ('pipeline' en inglés) con dos etapas (cargar y ejecutar), que les permite ejecutar la mayoría de operaciones en un ciclo de reloj, lo que los hace relativamente rápidos entre los microcontroladores de 8 bits.

El set de instrucciones de los AVR es más regular que la de la mayoría de los microcontroladores de 8 bit (por ejemplo, los PIC). Sin embargo, no es completamente ortogonal:

- Los registros punteros tienen capacidades de direccionamiento diferentes entre sí
- Los registros 0 al 15 tienen diferentes capacidades de direccionamiento que los registros 16 al 31, y los registros 32 al 63
- La instrucción CLR afecta los 'flag', mientras que la instrucción SER no lo hace, a pesar de que parecen ser instrucciones complementarias (dejar todos los bits en 1, y dejar todos los bits en 0 respectivamente).

Familias AVR

Los microcontroladores AVR se pueden clasificar dentro de cinco familias:

- **tinyAVR.** (serie “ATtiny”), de 0,5-8kB de memoria de programa, encapsulados de 6-32 pines y un número muy limitado de periféricos internos. De tamaño muy reducido
- **megaAVR.** (serie ATmega) de 4-256kB de memoria de programa, encapsulados de 28-100 pines y gran cantidad de periféricos y de instrucciones implícitas. De propósito general
- **XMEGA.** (serie ATxmega) de 16-384kB de memoria de programa, encapsulados de 44, 64 y 100 pines, gran cantidad de periféricos y de características de mejora del rendimiento. Para sistemas embebidos de relativa potencia.
- **AVR de aplicación específica.** otros modelos de megaAVR con características especiales, como controladores LCD, puertos USB, ethernet, etc...
- **Atmel At94k.** (Circuito integrado programable en campo a nivel de sistema “FPSLIC”). Basados en FPGAs y usan la SRAM como memoria de programa.

Recursos de información AVR

Como los PIC, tiene una comunidad de seguidores (ejemplificadas por el foro de internet AVRFreaks [9]), principalmente debido a la existencia de herramientas de desarrollo gratuitas o de bajo coste. Estos microcontroladores están soportados por tarjetas de desarrollo de costo razonable, capaces de descargar el código al microcontrolador, y por una versión de las herramientas GNU. Esto último es posible por su uniformidad en el acceso al espacio de memoria, propiedad de la que carecen los procesadores de memoria segmentada o por bancos, como el PIC o el 8051 y sus derivados.

En el anexo (apartado 7.2) se puede encontrar una breve descripción de los lenguajes y formas de programación de los microcontroladores

2.3. Sistema Operativo

Pese a que en este proyecto no se modificará en absoluto un sistema operativo, pues se utilizan estándares USB compatibles con cualquier sistema, resulta interesante resaltar las distintas capas de un sistema operativo, y así entender cómo trabaja un dispositivo de interfaz humana a la hora de comunicarse con la aplicación final

2.3.1. Introducción

"Un Sistema operativo (SO) es un software que actúa de interfaz entre los dispositivos de hardware y los programas usados por el usuario para manejar un ordenador. Es responsable de gestionar, coordinar las actividades y llevar a cabo el intercambio de los recursos y actúa como estación para las aplicaciones que se ejecutan en la máquina." Esto queda reflejado en la figura 10.

Figura 10. Abstracción de capas de un sistema informático

Es decir, por ejemplo, Windows de Microsoft es un programa escrito para controlar unos componentes físicos (o hardware) de ordenador (procesador, sistemas de vídeo, memoria RAM,...), atenderlos de manera conveniente según las distintas características, y que ofrezca una serie de herramientas para interactuar con el total del sistema. De esta forma se consigue que se puedan desarrollar aplicaciones en función de un sistema operativo con independencia del hardware que éste utilice.

Es, por tanto, un error común confundir las herramientas del sistema operativo, como por ejemplo el "Escritorio" de Windows o el "administrador de archivos" con el propio sistema operativo, pues son otras aplicaciones que se apoyan en el sistema operativo, que es sencillamente el núcleo o kernel. De tal forma que es posible cambiar radicalmente el aspecto y funcionalidad de un sistema completo sin cambiar el sistema operativo.

2.3.2. Historia.

Los primeros sistemas (1945 - 1950) eran grandes máquinas operadas desde la consola maestra por los programadores. Durante la década siguiente (1950 - 1960) se llevaron a cabo avances en el hardware: lectoras de tarjetas, impresoras, cintas magnéticas, etc. Esto a su vez provocó un avance en el software: compiladores, ensambladores, cargadores, controladores de dispositivos, etc.

A finales de los años 80, un Amiga equipado con una aceleradora "Video Toaster", era capaz de producir efectos comparados a sistemas dedicados que costaban el triple.

2.3.3. Funcionamiento dentro de un ordenador.

Al encender un ordenador se carga un pequeño programa escrito en lenguaje ensamblador, la BIOS. Almacenado en una memoria ROM, contiene la configuración de los elementos hardware básicos del PC y se encarga de cargar el sistema operativo en la memoria RAM.

El núcleo actúa de intermediario entre las aplicaciones y el hardware. El núcleo tiene como funciones básicas garantizar la carga y la ejecución de los procesos, las entradas/salidas y proponer un interfaz entre el espacio del núcleo y los programas del espacio del usuario. De esta forma, para que una aplicación sea funcional en un sistema operativo, ésta le tiene que solicitar un espacio en memoria y acceso a otros recursos.

Una vez cargado, los sistemas operativos actuales, cargan a su vez los controladores de periféricos ("Drivers") no incluidos en el núcleo y las herramientas del sistema operativo, así como otros programas configurados en el inicio.

Un sistema operativo implica un gran número de conceptos, sin embargo los más reseñables relacionados con la comunicación con un dispositivo externo, motivo de estudio de este proyecto, son:

- **Sistema de E/S.** Consiste en un sistema de almacenamiento temporal ("caché"), una interfaz de controladores de dispositivos y otra para dispositivos concretos. El sistema operativo debe gestionar el almacenamiento temporal de E/S y servir las interrupciones de los dispositivos de E/S y que estas no interfieran de manera directa con la ejecución del procesador
- **HAL** (siglas inglesas de Hardware Abstraction Layer) o Capa de abstracción de hardware es una aplicación que está constantemente funcionando de manera transparente al usuario ("demonio"), que permite a las aplicaciones un fácil acceso a la información sobre el hardware de manera que puedan localizar y utilizar tal hardware independientemente del tipo de bus o dispositivo. De esta manera una Interfaz gráfica de usuario puede presentar a su usuario todos los recursos de una manera uniforme y completa.

La capa HAL, por ejemplo, puede recoger información acerca de los dispositivos de almacenamiento removibles y crear un icono en el escritorio del usuario, permitiendo así, facilidades de acceso y modificación, así como relativa independencia del hardware

2.3.4. Drivers

Un controlador de dispositivo, llamado normalmente controlador (en inglés, device driver, o simplemente driver) es un programa informático que permite al sistema operativo interactuar con un periférico, haciendo una abstracción del hardware y proporcionando una interfaz para usarlo sin necesidad de saber cómo funciona internamente el periférico. Conceptualmente se puede establecer una analogía a un manual de instrucciones que le indica al sistema operativo cómo debe controlar y comunicarse con un dispositivo en particular. Por tanto, es una pieza esencial, sin la cual no se podría usar el hardware.

Es importante resaltar que el software de controladores de dispositivos se ejecuta como parte del sistema operativo, con acceso sin restricciones a todo el equipo, por lo que resulta esencial que el controlador de dispositivo sea de confianza.

Pueden ser de dos tipos: orientados a caracteres o bien orientados a bloques, constituyendo las conocidas unidades de disco. La diferencia fundamental entre ambos tipos de controladores es que los primeros reciben o envían la información carácter a carácter; en cambio, los controladores de dispositivo de bloques procesan, como su propio nombre indica, bloques de cierta longitud en bytes (sectores).

2.4. Software Libre

2.4.1. Filosofía

Un software comercial es un producto, por así decirlo, manufacturado, más concretamente compilado. Un programador escribe un programa con un lenguaje de alto nivel, relativamente fácil de entender y seguir por otro programador. Luego lo "compila" con otro software, que lo traduce a código inteligible por el procesador y se generan los archivos binarios ejecutables.

Estos binarios funcionan como una caja negra a efectos del usuario final: actúa con él y obtiene salidas, pero sin llegar a saber que está haciendo realmente. Este código generado ya no es traducible al lenguaje de alto nivel en el que se escribió, tan sólo a un código similar al ensamblador. Se pierde, como mínimo, la perspectiva funcional del código fuente, haciendo casi imposible realizar cambios importantes. Es una caja negra.

Cuando un usuario compra un software, generalmente está pagando por los binarios. Es un concepto perfectamente justificable, pues al igual de otros modelos de negocio, la materia prima son ideas plasmadas en lenguaje escrito y la manufactura la ejecuta una aplicación relativamente asequible e incluso gratuita. No se puede negar lo beneficioso de este modelo económico para el mundo, pues ha supuesto una revolución que en tan sólo 30 años casi la mitad de los hogares del mundo posean un PC.

Sin embargo, el concepto de "caja negra" crea ciertos inconvenientes.

- No se puede saber qué se está ejecutando y cómo, de esto se derivan, por ejemplo, quejas de usuarios por software inflado ("Bloatware"). Ya que cada vez se necesitan mayores recursos computacionales para ejecutar versiones más modernas de programas que no aportan nuevas funcionalidades.
- No se puede ajustar a las necesidades o cambiar funcionalidades si no está preparado para ello. Esto hace que, por ejemplo PYMEs, tengan que adaptarse a un software por no poder permitirse una modificación por parte de la empresa desarrolladora.
- Se distancia cada vez más al usuario de la técnica. En otras áreas el usuario está en contacto con la tecnología que emplea. En el área de la automoción, por ejemplo, un conductor de un automóvil con conocimientos básicos de mecánica sabe perfectamente que, cuando se ha pinchado una rueda, debe cambiarla. En cambio, con el software comercial, no es tan trivial dar con la solución de problemas sencillos.

El software libre es exactamente lo contrario. Según la Free Software Foundation, el software libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software; de modo más preciso, se refiere a cuatro libertades de los usuarios del software:

- la libertad de usar el programa, con cualquier propósito
- de estudiar el funcionamiento del programa
- de adaptarlo a las propias necesidades
- de distribuir copias, con lo cual se puede ayudar a otros y de mejorar el programa y hacer públicas las mejoras, de modo que toda la comunidad se beneficie (para la segunda y última libertad mencionadas, el acceso al código fuente es un requisito previo).

En la figura 11 se representa el mapa de las implicaciones inherentes al software libre. Es importante resaltar los valores de intrínsecos de esta forma de creación, pues son por definición altruistas, a la vez que se muestra perfectamente compatible con otros modelos de negocio, como el soporte.

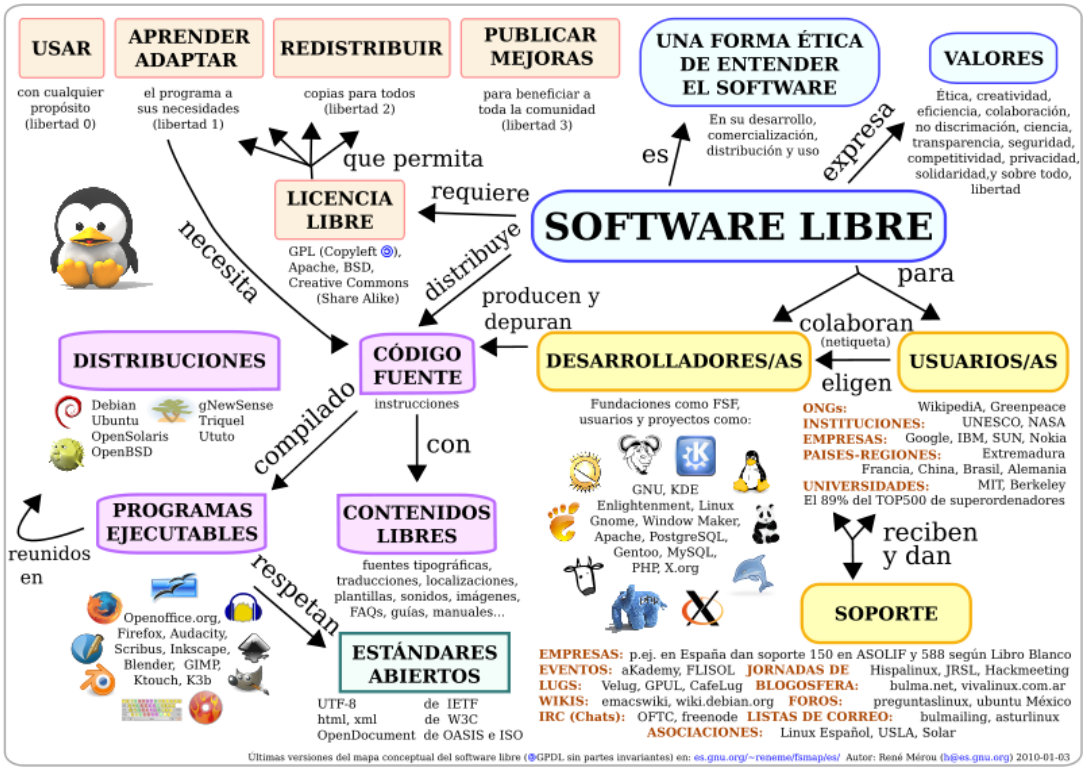

Figura 11. Diagrama de conceptos del software libre

2.4.2. Historia

A principios de los 70, cuando la informática todavía no había sufrido su gran boom, las personas que hacían uso de ella, en ámbitos universitarios y empresariales, creaban y compartían el software sin ningún tipo de restricciones. Pero con la llegada de los años 80 la situación empezó a cambiar. Los ordenadores más modernos comenzaban a utilizar sistemas operativos privativos, forzando a los usuarios a aceptar condiciones restrictivas que impedían realizar modificaciones a dicho software.

Se cuenta que Richard Stallman trabajaba en un laboratorio que tenía una impresora que causaba frecuentes pérdidas de tiempo debido a un error en su diseño de controlador, y que cuando fue a pedir el código fuente a la empresa vendedora para solucionar el problema se lo negaron. Y es cuando se vio en la encrucijada de trabajar con software privativo para generar más software privativo o romper aquel círculo.

Con este antecedente, en 1984, Richard Stallman comenzó a trabajar en el proyecto GNU, y un año más tarde fundó la Free Software Foundation (FSF). Stallman introdujo la definición de free software y el concepto de "copyleft", que desarrolló para otorgar libertad a los usuarios y para restringir las posibilidades de apropiación del software.

2.4.3. Licencias

Existen varias licencias que intentan respetar las ideas del software libre con distintas peculiaridades y usos. No confundir con “software gratis” (o freeware) pues la gratuidad no implica la capacidad de modificarlo o distribuirlo. Las más comunes son:

a. GNU GPL(Licencia pública general GNU):

Es una de las más utilizadas, el autor conserva los derechos de autor (copyright), y permite la redistribución y modificación bajo términos diseñados para asegurarse de que todas las versiones modificadas del software permanecen bajo los términos más restrictivos de la propia GNU GPL. Esto hace que sea imposible crear un producto con partes no licenciadas GPL: el conjunto tiene que ser GPL.

Es decir, la licencia GNU GPL posibilita la modificación y redistribución del software, pero únicamente bajo esa misma licencia. Y añade que si se reutiliza en un mismo programa, un código "A" licenciado bajo licencia GNU GPL y un código "B" licenciado bajo otro tipo de licencia libre, el código final "C", independientemente de la cantidad y calidad de cada uno de los códigos "A" y "B", deberá estar bajo la licencia GNU GPL.

En la práctica esto hace que las licencias de software libre se dividan en dos grandes grupos, aquellas que pueden ser mezcladas con código licenciado bajo GNU GPL (y que inevitablemente desaparecerán en el proceso, al ser el código resultante licenciado bajo GNU GPL), y las que no lo permiten. Estas últimas, al incluir mayores u otros requisitos que no contemplan ni admiten la GNU GPL no pueden ser enlazadas ni mezcladas con código gobernado por la licencia GNU GPL.

Figura 12. Logotipo de licencia GNU

b. Tipo BSD.

Compatibles con las licencias GNU GPL pues se basan en los mismos principios pero ésta resulta más permisiva pues permite que se re-licencie a cualquier otra licencia, incluso privativas.

Stallman establece una analogía sobre su relativa mayor permisividad que las licencias GPL, "una licencia BSD es más libre que una GPL si y sólo si se opina también que un país que permita la esclavitud es más libre que otro que no la permite"

c. Creative Commons.

Es una licencia inspirada en las licencias GPL, pero a diferencia de ésta, no es aplicable para programas informáticos, sino para contenidos creativos, como literatura, música, imágenes,... .

Figura 13. Logotipos identificativos de licencia Creative Commons

Por ejemplo, este proyecto está licenciado bajo licencia Creative Commons (BY, NC), lo que significa que puede ser utilizada de cualquier forma y con cualquier fin, siempre que no sea comercial y se reconozca al autor (figura 13).

2.4.4. Regulación

En España la Orden EDU/2341/2009, de 27 de agosto, por la que se crea el Centro Nacional de Desarrollo Curricular en Sistemas no Propietarios, tiene como finalidad el diseño, el desarrollo y la promoción de contenidos educativos digitales para colectivos educativos específicos, en el ámbito de las Tecnologías de la Información y la Comunicación, que se centra en promocionar y aplicar estrategias dirigidas a poner a disposición de los centros escolares recursos y contenidos digitales de calidad, desarrollados en software libre.

2.4.5. Problemas

El software libre presenta ciertos problemas inherentes a su filosofía. El software se distribuye sin garantía de ningún tipo, el autor no se responsabiliza de los daños que se puedan causar al sistema por utilizar su software o de si será utilizable.

Además, pese a que cada vez se trabaja más para paliar este defecto, por lo general se requerirán mayores conocimientos para utilizar un programa libre que para uno similar privativo. Y es de entender pues, el autor del programa busca ante todo la estabilidad y funcionalidad del software y no mejorar su cosmética o documentación.

También se habla del concepto bazar frente al de catedral, a partir de una controvertida obra de Eric S. Raymond, como las dos formas de afrontar el desarrollo libre. El concepto catedral implica un diseño estructurado desde la incepción de un gran proyecto, en cambio el concepto bazar consiste en ir desarrollando en función de lo que se necesita hasta conseguir completar el proyecto. Las críticas a esta discusión proceden de la simplificación del problema, aunque para los propósitos de este proyecto resulta clarificador

2.4.6. Desarrollo distribuido

Puesto que una de las virtudes del software libre es que puede ser desarrollado por cualquier persona interesada, el código fuente se acostumbra a publicar en un sistema de control de versiones. Esta aplicación, cliente-servidor, crea versiones a partir de cada modificación que se aplica al código. De tal forma que si, por ejemplo, una aplicación se encuentra en su revisión 30, un colaborador puede proponer al administrador una mejora y que el responsable del software la publique en una nueva revisión, la 31, que modificará la totalidad del código.

Existen varios software de control de versiones, sin embargo los más destacados son, el original (“CVS”), el Subversion (“SVN”), que corregía ciertas limitaciones; y el más actual y avanzado “GIT”, diseñado por Linus Torvalds que crea un nuevo concepto de programación distribuida.

2.5. GNU/Linux

GNU/Linux es uno de los términos empleados para referirse a la combinación del núcleo o kernel libre similar a Unix denominado Linux, que es usado con herramientas de sistema GNU.

Su desarrollo es uno de los ejemplos más prominentes de software libre, pues se trata de un sistema operativo completo y funcional, en el que todo su código fuente puede ser utilizado, modificado y redistribuido libremente bajo los términos de la GPL (Licencia Pública General de GNU) y otra serie de licencias libres.

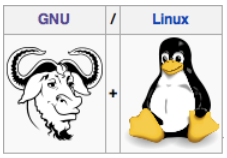

Figura 14. Esquema de GNU/Linux

2.5.1. Historia

El proyecto GNU, iniciado en 1983 por Richard Stallman, tiene como objetivo el desarrollo de un sistema operativo Unix completo compuesto enteramente de software libre. La historia del núcleo Linux está fuertemente vinculada a la del proyecto GNU. En 1991 Linus Torvalds empezó a trabajar en un reemplazo no comercial para MINIX que más adelante acabaría siendo Linux.

Cuando Torvalds liberó la primera versión de Linux, el proyecto GNU ya había producido varias de las herramientas fundamentales para el manejo del sistema operativo, incluyendo un intérprete de comandos, una biblioteca C y un compilador. Pero pese a que el proyecto contaba con una infraestructura para crear su propio sistema operativo, llamado Hurd, este aún no era lo suficiente maduro para usarse y comenzaron a usar a Linux a modo de continuar desarrollando el proyecto GNU. Entonces, el núcleo creado por Linus Torvalds, quien se encontraba por entonces estudiando en la Universidad de Helsinki, llenó el "espacio" que había en el sistema operativo de GNU.

El día en que se estime que Hurd es suficiente maduro y estable, será llamado a reemplazar a Linux. Sin embargo dada la extremada especificidad de este núcleo y el extenso abanico de sistemas, puede ser que ese día sea inalcanzable.

En la figura 15 se puede ver el uso de Linux en 2008 en ordenadores personales. En este gráfico no se contemplan servidores, dónde el uso de GNU/Linux está más extendido.

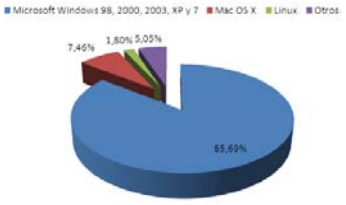

Figura 15. Uso actual de sistemas operativos (datos orientativos)

2.5.2. Funcionalidad y aspecto.

En la actualidad existen muchas variantes de linux, conocidas como “sabores”. Y la mayoría de ellas tienen, a su vez, versiones para servidores (profesionales) y para ordenadores personales (de escritorio). Las versiones para escritorios están pensadas para usuarios que no, necesariamente, tengan conocimientos avanzados y presentan un entorno y funcionalidad similar a los de los sistemas Windows de Microsoft.

Tienen escritorio y un sistema de archivos en árbol (figura 16). Navegadores de internet, generalmente Firefox. Reproductores de vídeos, música. Editores de imágenes, de textos, hojas de cálculo, bases de datos. Estas aplicaciones, pueden no ser las mismas de otros sistemas privativos, pero trabajan sobre los mismos archivos y formatos, lo que permite una compatibilidad total en el trabajo realizado.

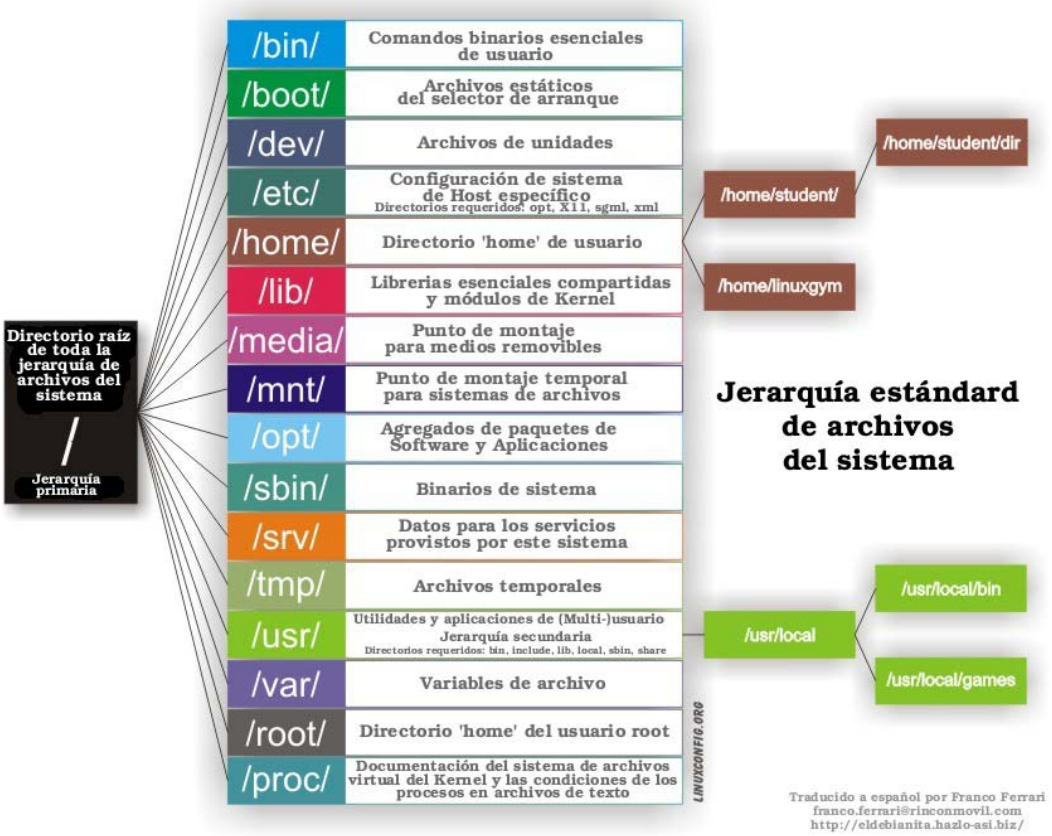

Figura 16. Estructura de arbol típica de GNU/Linux

Además, estas aplicaciones suelen incluirse en la instalación y presenta una peculiaridad a la hora de instalar nuevas aplicaciones con respecto a otros sistemas, como Windows. Son los conocidos como repositorios, que son un índice de aplicaciones probadas y certificadas, listas para ser descargadas e instaladas. Además, claro ésta, de la tradicional forma de conseguir el paquete de instalación de la versión correspondiente o de compilar desde el código fuente.

2.5.3. Visión general de GNU/Linux en función del ROL

En la figura 17 se muestra un interesante esquema de la anatomía de Linux desde el punto de vista de los roles más significativos de GNU/Linux. En él se puede apreciar la elegancia de la estructura lógica de este sistema operativo. Se puede entrever como en la conjunción GNU/Linux concurren todos las soluciones a las necesidades de cualquier usuario

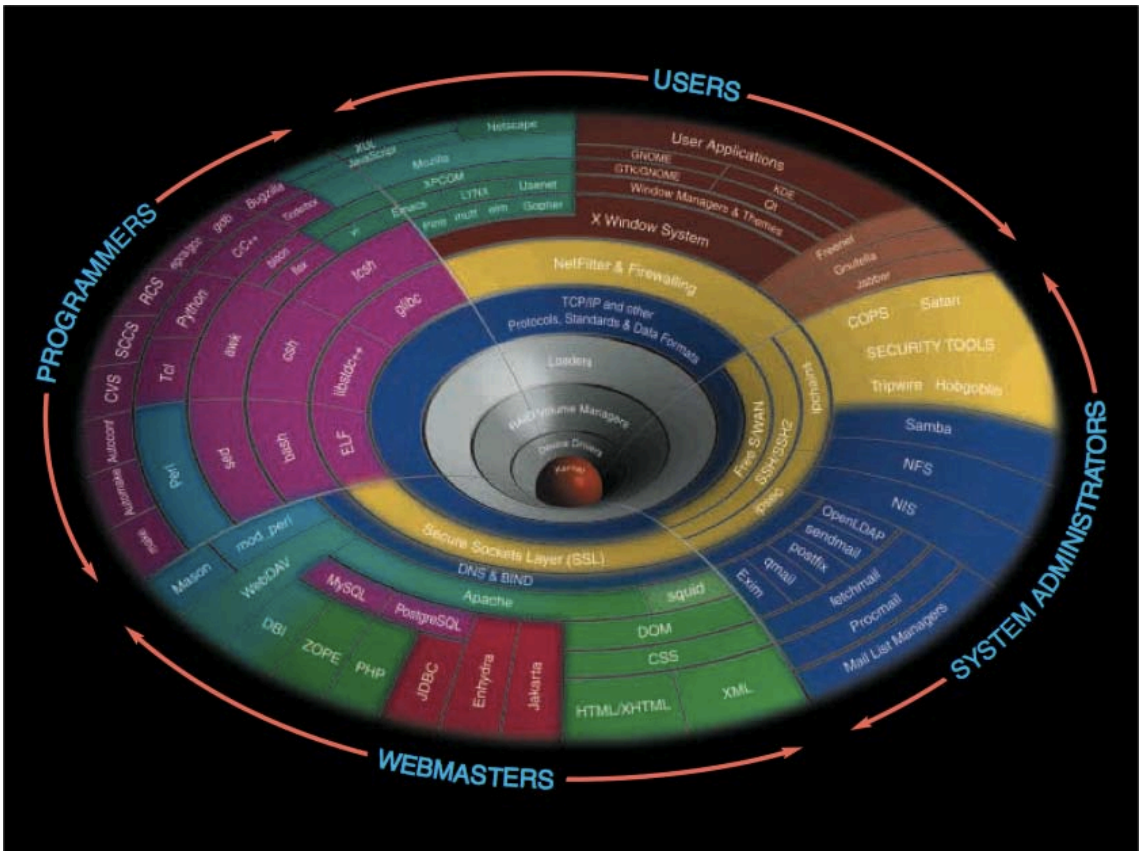

Figura 17. Universo Linux según publicación O'Relly

2.6. Ubuntu

2.6.1. Introducción

Ubuntu es una distribución Linux basada en Debian GNU/Linux que proporciona un sistema operativo actualizado y estable para el usuario medio, con un fuerte enfoque en la facilidad de uso y de instalación del sistema. Al igual que otras distribuciones se compone de múltiples paquetes de software normalmente distribuidos bajo una licencia libre o de código abierto.

Está patrocinado por Canonical Ltd., una compañía británica propiedad del empresario sudafricano Mark Shuttleworth, que se financia por medio de servicios vinculados al sistema operativo.

Figura 18. Logotipo de Ubuntu

Cada seis meses se publica una nueva versión de Ubuntu, la cual recibe soporte por parte de Canonical durante dieciocho meses, por medio de actualizaciones de seguridad, parches para bugs críticos y actualizaciones menores de programas. Las versiones LTS (Long Term Support), que se liberan cada dos años, reciben soporte durante tres años en los sistemas de escritorio y cinco para la edición orientada a servidores.

2.6.2. Uso

Pese a ser duramente criticado por la FSF de Stallman por incluir software privativo y no ser la distribución de mayores prestaciones o características, sí que es la más sencilla de instalar, usar y de documentarse en internet, por lo que resulta la más recomendada para los usuarios que se inician en Linux. De hecho todo el software utilizado para el desarrollo de este proyecto se ha utilizado a partir de esta distribución, sin necesidad de conocimientos específicos.

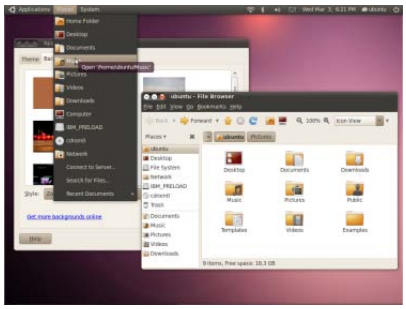

Figura 19. Captura de pantalla de Ubuntu

La forma más sencilla de probar este sistema operativo consiste en descargar desde su página oficial la versión más actual que se acomode al sistema de destino. Grabar esta imagen en un CD y arrancar el sistema con él dentro de la bandeja principal de CDs. Ejecutar el modo “live” y familiarizarse con el entorno. Pese a que esta forma de probar el S.O. es totalmente funcional, todos los cambios e instalaciones que se realicen en el sistema no perdurarán tras un reinicio del sistema, sin embargo, a partir de aquí también es posible instalar el SO. Como es habitual

tener un sistema de Microsoft instalado en el sistema, para no perder los datos ni esa instalación, el gestor de instalación, ofrece la posibilidad de combinar los dos sistemas operativos y elegir cual arrancar al iniciar el PC.

2.6.3. Synaptic

Es una aplicación para la representación gráfica de "apt" (el repositorio de Debian), el gestor de paquetes del sistema Ubuntu. Combina la simplicidad de apuntar y clickear (figura 20) con la potencia del comando apt-get en la línea de comandos. Permite, como se puede apreciar (en parte) en la figura 20:

- Instalar, eliminar, configurar y actualizar paquetes software.
- Buscar, listar y organizar una lista de los paquetes software disponibles.
- Gestionar repositorios y actualizar el sistema completo.

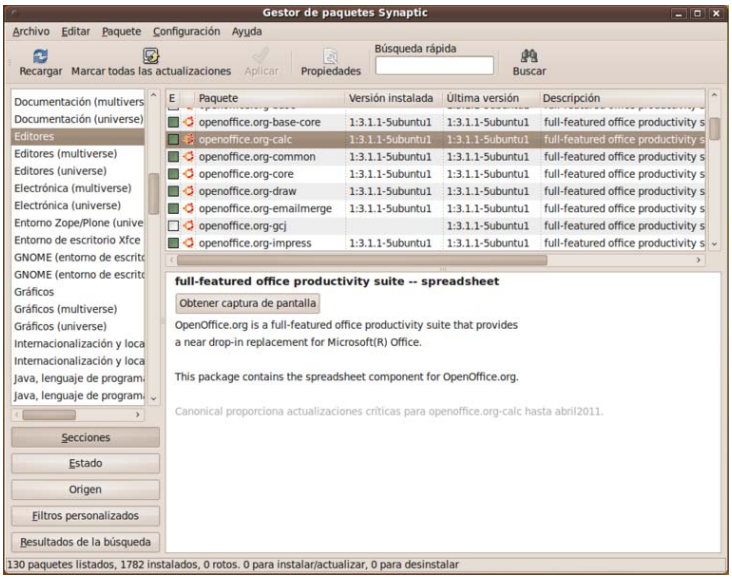

Figura 20. Captura de pantalla Synaptic

Puesto que encontrar la versión apropiada para el sistema, de un determinado software, puede no ser una tarea fácil, esta aplicación se encarga de listarla si está disponible y de listar además posibles paquetes requeridos para su instalación. Si por ejemplo se quisiera instalar el paquete ofimático "openoffice.org", habría que buscarlo en la barra de búsqueda y marcar el paquete deseado. A continuación, al clickear en instalar, la ventana emergente de confirmación marcaría otros paquetes necesarios para la instalación.

Una vez finalizada la descarga y posterior instalación, el programa estará configurado para su uso y ubicado en la barra de herramientas para que el usuario lo pueda encontrar con sencillez.

Una vez más cabe decir, que todo el software utilizado para la realización de este proyecto ha sido instalado a partir de este interfaz.

2.7. Fritzing

Es un programa basado en la licencia libre gpl v3, para el diseño de placas electrónicas. Se basa en un sistema de dibujo sencillo que permite conectar elementos electrónicos prediseñados, así como otros diseñados por el usuario, y ofrecer vistas esquemáticas, para la inserción en protoboard e incluso de impresión sobre PCB (figura 22).

Figura 21. Logotipo de Fritzing

Es el software elegido para ilustrar los ejemplos y esquemas de este proyecto, y está disponible en los repositorios de Ubuntu

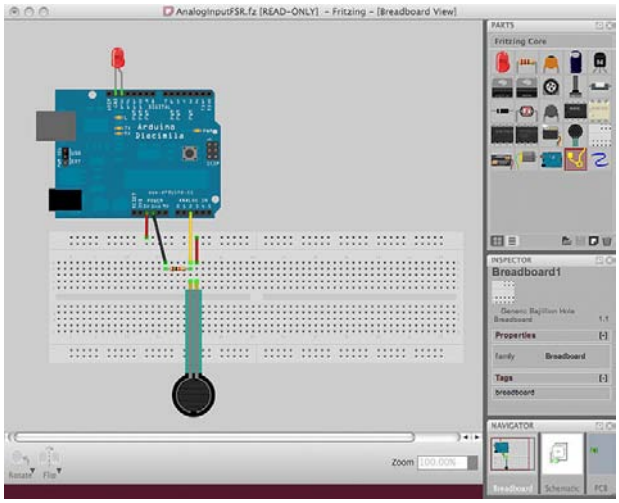

Figura 22. Captura de pantalla de Fritzing

2.8. Secret Maryo Chronicles

Una de las principales fuentes de inspiración a la hora de desarrollar proyectos libres, es evidentemente, el software privativo. Cuando trasciende la inspiración y se busca ofrecer una experiencia lo más similar posible, se dice que se hace un “clon libre”.

En este caso el “Secret Maryo Chronicles” es un clon libre del “Super Mario Bros” de ”Nintendo”, al que han tenido que modificar determinadas imágenes de marca para no incurrir en delito contra la propiedad intelectual (figura 23).

Además dada su naturaleza libre, presenta ciertas ventajas con respecto al software privativo en el que se basa, pues la comunidad puede contribuir con mapas y modificaciones del sistema de juego, que pueden hacer la experiencia más rica.

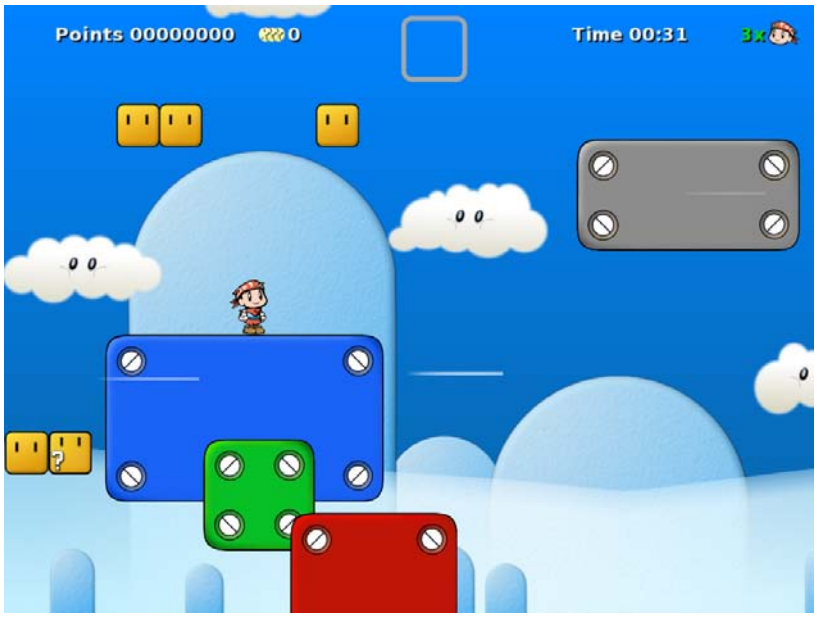

Figura 23. Captura de pantalla de Secret Maryo Chronicles

Se ha elegido este software como banco de pruebas para el HID-USB que se desarrolla en este proyecto, pues el propio juego está diseñado para interpretar las entradas (framebuffer más específicamente) de una amplia variedad de Joystick, entre ellos, el genérico utilizado.

2.9. Generalidades USB

Una vez estudiados todos los sistemas relacionados con el proyecto desde una perspectiva generalista, en los siguientes apartados del estado de la técnica, se detallará el funcionamiento y uso del protocolo de comunicaciones utilizado, el USB ("Universal Serial Bus").

Aunque se intentará, en la medida de lo posible, seguir la tónica del proyecto y presentarlo de una manera sencilla, los siguientes apartados es posible que requieran de un conocimiento técnico específico para su comprensión.

2.9.1. Historia

Nace para acabar con la diversidad de conectores para acceder al PC, más sencillo y con más prestaciones, en especial las multimedia. La alianza de varios fabricantes (Intel, Compaq, NEC y Microsoft) fija como objetivo lograr un bus con anchos de banda suficientes para la conexión de dispositivos diferentes sin reestructurar el hardware ni cargar software especifico, es decir, un conector externo, de un coste no muy elevado, para todos los dispositivos actuales y futuros.

Para definir los principios se establecieron los siguientes criterios:

- Exige el cumplimiento del estándar. (estándar no muy restrictivo).
- Permite su conexión en caliente, con plug&play, y proporcionar alimentación para el dispositivo que se conecte.
- Cableado sencillo y universalista: Implementado de modo sencillo sobre un conector único pero universalista, es decir, válido para cualquier dispositivo.
- De coste bajo sin arriesgar la velocidad.
- Diversidad de interconexiones: Interconecta dispositivos de voz, audio y video comprimido.
- Versatilidad en protocolos: Transfiere en modo sincrono audio y video, y en asíncrono, mensajes.
- Arquitectura abierta a nuevas clases de periféricos: Arquitectura básica abierta para toda clase de periféricos y facilidad de interconectar dispositivos a concentradores formando topologías de estrella con concentradores en cascada.

A diferencia de otros interfaces, USB no asigna funciones específicas a líneas de señales o hace asunciones acerca de qué interfaz se usará. Por ejemplo, en el puerto paralelo hay cinco entradas/conexiones que las impresoras utilizan para indicar su estado. Para el USB, no hay conexiones ("cables") dedicados, hay clases ("lógicas") definidas que especifican los requerimientos y el protocolo a utilizar, en función del dispositivo que represente.

2.9.2. Cronología

USB 0.9 (1995): Primer borrador.

USB 1.0 (1996): Establece dos tipos de conexión:

- Velocidad baja ("Low speed") de 1,5 Mbps y pensada para periféricos de pequeño ancho de banda, como ratones o joysticks.
- Velocidad “completa” ("Full speed") de 12 Mbps, destinada a los dispositivos más rápidos.

USB 1.1 (1998): Añade detalles y precisiones a la norma inicial. Es el estándar mínimo que debe cumplir un dispositivo USB.

USB 2.0 (2000): Extensión de la norma compatible con las anteriores. Con velocidades de 480 Mbps(alta velocidad o "High speed").

USB 3.0 (En la actualidad): Con velocidades de hasta 4.8Gb/s

2.9.3. El conector

Presenta muchas formas estándar (figura 24), y otros fabricantes han adaptado su conector para sus aplicaciones, no obstante, siempre se compone de 4 cables. Las señales del USB se transmiten en un cable de par trenzado con impedancia característica de 90 Ω ± 15%, cuyos hilos se denominan D+ y D-. Estos, colectivamente, utilizan señalización diferencial en full dúplex para combatir los efectos del ruido electromagnético en enlaces largos.

D+ y D- suelen operar en conjunto y no son conexiones simples. Los niveles de transmisión de la señal varían de 0 a 0'3 V para bajos (ceros) y de 2'8 a 3'6 V para altos (unos) en las versiones 1.0 y 1.1, y en ±400 mV en alta velocidad (2.0). En las primeras versiones, los alambres de los cables no están conectados a masa, pero en el modo de alta velocidad se tiene una terminación de 45 Ω a tierra o un diferencial de 90 Ω para acoplar la impedancia del cable.

Figura 24. Distintos conectores estándar (hasta USB 2.0)

2.9.4. Arquitectura

Un periférico USB es un dispositivo inteligente que sabe cómo responder a las peticiones y eventos del bus de datos. Es decir, con un USB no se puede simplemente leer y escribir en la dirección física de un puerto, el dispositivo debe habilitar un protocolo para que un PC lo detecte, lo identifique y se comunique con él.

De esta forma se puede utilizar una topología de estrella en cascada (figura 25), siendo los HUB, o repartidores, los centros de las estrellas, los periféricos, las puntas. Y claro está, un USB anfitrión que controla toda esa línea de datos (o buses). Se puede tener conectado hasta un total de 127 dispositivos con 7 “peldaños” (o tiers) de hub.

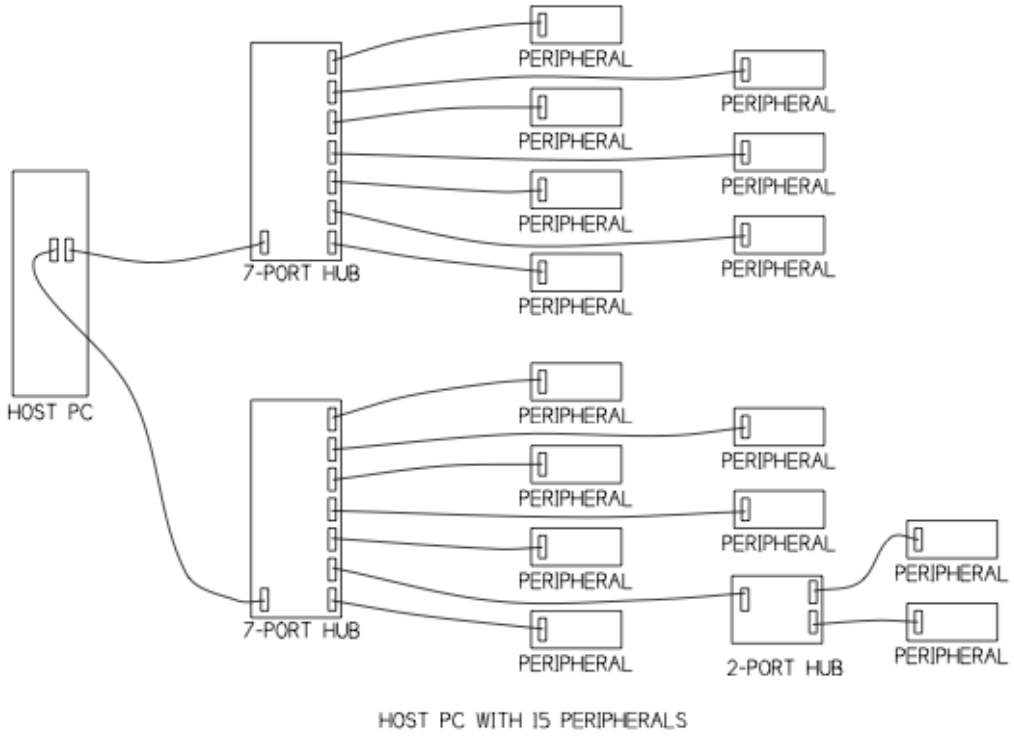

Figura 25: Ejemplo de topología de estrella en cascada, a partir de un PC con dos controladoras de anfitriones USB (USB Complete Third Edition)

Para ejecutar esta topología se divide la labor y se definen responsabilidades tanto para el anfitrión (Host) o el dispositivo (Peripheral).

2.9.5. Anfitrión:

Está a cargo del bus, debe saber qué dispositivos están en el bus y sus capacidades, además se debe asegurar que todos los dispositivos pueden recibir y enviar los datos que necesiten.

Para el caso de un ordenador que actúe como anfitrión USB, el hardware y el sistema operativo, a través de los "drivers", llevan toda la carga del bus y la aplicación informática no debe preocuparse de los detalles específicos de la comunicación con el USB.

El anfitrión por tanto lleva a cabo las siguientes tareas:

- **Detectar dispositivos**: En el encendido, el "Hub" informa de todos los dispositivos conectados. En un proceso llamado "enumeración" el anfitrión asigna direcciones y solicita información adicional de cada dispositivo. Si un dispositivo es apagado o conectado, el anfitrión detecta el evento y lo elimina de la lista o enumera nuevamente, según el caso.
- **Gestionar el flujo de datos**: Varios periféricos pueden querer transmitir datos al mismo tiempo. El anfitrión divide, por tanto, el tiempo disponible en segmentos y los asigna en función de las especificaciones recabadas durante la enumeración.
- **Comprobación de errores**: En la transmisión de datos, el anfitrión añade bits para la comprobación de errores. Si en el dato recibido por el periférico no se corresponde el bit de comprobación de error con su cálculo, éste no enviará la señal de reconocimiento ("acknowledge") y el anfitrión sabrá que deberá reenviar la información. Existe también, una variante que no permite la re-transmisión para mantener el ratio de transmisión constante.
- Además el anfitrión puede recibir **otras indicaciones** de los periféricos, como que no puede enviar o recibir datos, e informar así al controlador del dispositivo sobre el problema.
- **Suministro de energía**: El cable USB utiliza una pareja de sus cables para suministrar +5V de tensión continua. En función de si se trata de baja o alta potencia, puede suministrar desde 100mA hasta 500mA respectivamente.
- **Intercambio de datos con los periféricos**: La tarea principal del anfitrión, puede comunicarse con los periféricos en los intervalos de tiempo o estar a la espera de solicitudes.

2.9.6. Periféricos:

A excepción de su función al conectarse, que informa al anfitrión para que le "enumere", el periférico debe esperar a que el anfitrión se comunique con él, para responder.

El controlador USB en un periférico maneja diversas responsabilidades del hardware en función del chip.

- **Detectar comunicaciones dirigidas al chip:** El dispositivo monitoriza la dirección de dispositivo en cada comunicación en el bus. Si no coincide con su dirección asignada, ignora la comunicación, si coincide, almacena esta información en el "buffer de recepción" y habilita la interrupción que indica que ha llegado información. En la mayoría de los chips esta función es realizada por el "hardware" y no necesita ser programada en el "firmware".
- **Responder a solicitudes estándar**: Cuando el dispositivo se conecta a un bus USB, debe responder a determinadas solicitudes durante la "enumeración". La especificación USB define 11 solicitudes, tales como capacidades, status y configuración. Sin embargo "vendor" (fabricante) puede, a su vez, definir solicitudes adicionales.
- **Comprobación de errores**: Al igual que el anfitrión, el dispositivo añade un bit de comprobación de error. Si el dispositivo no confirma la información, el anfitrión sabrá que deberá reenviar la información. Esta función suele estar implementada en el hardware, por lo que no es necesario programarla en el "firmware".
- **Gestión de energía**: El dispositivo puede ser alimentado por el bus, sujeto a límites, o puede tener su propia fuente de alimentación. Si el anfitrión entra en un estado de baja potencia, cesan las comunicaciones. Si se mantiene durante, al menos, 3 milisegundos, el dispositivo deberá entrar en estado de suspensión.
- **Intercambio de datos con el anfitrión**: Tras la "enumeración", el dispositivo queda configurado para el anfitrión y esté establecerá que tipo de comunicación correspondiente, y cuándo ocurre. Si el anfitrión envía datos al dispositivo, el dispositivo debe responder a cada intento de transferencia enviando un código de que se acepta la transmisión o de que está ocupado. Si el dispositivo envía datos al anfitrión, el dispositivo debe responder cada intento del anfitrión con los datos a enviar, con un indicador de que no hay datos que enviar o de que está ocupado.

2.10. Transferencia USB

2.10.1. Elementos de una transmisión:

La comunicación se basa en el intercambio de paquetes de información, y para entender este intercambio se deben definir dos conceptos: los puntos finales (“endpoints”) y las tuberías (“pipes”).

a. Dispositivos puntos finales: la fuente y el sumidero de datos,

Todo el tráfico de un bus empieza o acaba en un dispositivo punto final (“device endpoint”). Siendo el punto final un bloque de memoria del chip del periférico que hace las veces de “buffer” para almacenar los datos recibidos o para los datos que esperan ser enviados. El anfitrión también almacena, de la misma forma, los datos. Sin embargo no posee puntos finales, pues sirve como iniciador y finalizador de la comunicación.

Una explicación sencilla sería equiparar el punto final a un buzón de correos, el cartero virtual, que representa al anfitrión, además de la dirección del dispositivo, debe saber a qué se corresponde cada buzón, si es para recoger envíos, o es para dejar la correspondencia, además debe saber de qué tipo de envío se trata, si es de control, o si es de datos, de qué tipo de datos.

Una dirección punto final consiste en un número, con un valor entre 0 y 15, y un código de dirección. Este código de dirección queda definido desde la perspectiva del anfitrión: “IN”, si envía información u “OUT”, si almacena información del anfitrión. En el caso de un punto final configurado como control, debe mantener una comunicación bidireccional, que consistiría en una pareja “IN” y “OUT” de códigos de direcciones que comparten el mismo número de punto final. Existe además otro código, “SETUP”, que hace se comporta como “OUT” especial y sirve para configurar el dispositivo. De esta forma cada transacción en el bus empieza con un paquete que contiene un número y un código de dirección.

Por tanto, cada transmisión contiene una dirección de dispositivo y una dirección de punto final. Si el dispositivo, tras comprobar la dirección del dispositivo, recibe un “OUT”, almacena esta información y habilita una interrupción, si recibe un “IN” y tiene la información lista, envía los datos y habilita una interrupción en el bus.

b. Tuberías:

Antes de que la transferencia ocurra, el anfitrión y el dispositivo deben establecer una tubería. Ésta se establece durante la "enumeración" y consiste en la asociación lógica entre el punto final del dispositivo específico y el software del controlador anfitrión.

El anfitrión, a su vez, puede crear o eliminar tuberías según la información tomada del dispositivo. Véase que todos los dispositivos tienen una tubería de control por defecto que usa el punto final 0.

Tipos de tuberías:

- **Tuberías de mensajes("Message Pipe"):** bidireccionales y se usan para la configuración

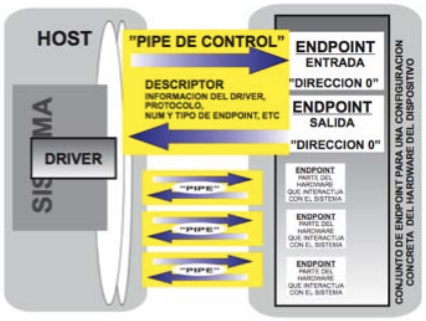

Figura 26. Esquema de tubería de mensaje/control

- **Tuberías de flujo ("Stream Pipe")**: unidireccionales y se usan para el resto de comunicaciones: transmisiones Generalistas ("Bulk"), Isócronas y de Interrupción.

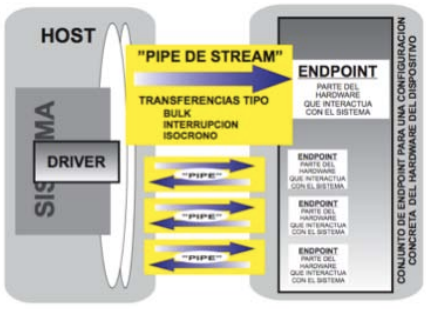

Figura 27. Esquema de tubería de flujo

2.10.2. Transferencia en detalle:

La transferencia (“transfer”) USB consiste en una serie de transacciones (“transaction”). Cada transacción tiene tres fases, o paquetes (“packet”), que ocurren en el siguiente orden: testigo (“token”), datos (“data”) y aceptación (“handshake”), figura 28.

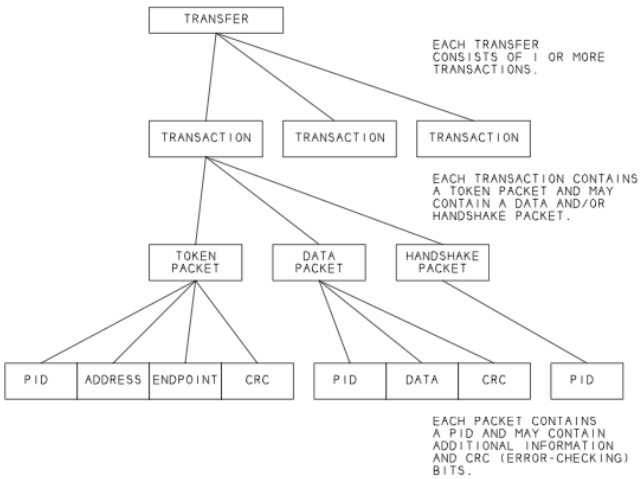

Figure 2-3: A USB transfer consists of transactions. The transactions in turn contain packets, and the packets contain a packet identifier (PID), PID-check bits, and sometimes additional information.

Figura 28. Detalle de paquetes y jerarquía (USB Complete Third Edition)

A su vez, cada fase consiste en varios paquetes enviados, pero siempre empiezan con el paquete “identificador de paquetes” (“PID”). Una referencia de estos identificadores se puede ver en la figura 29.

Packet Type	PID Name	Value	Transfer types used in	Source	Bus Speed	Description
Token (identifies transaction type)	OUT	0001	all	host	all	Device and endpoint address for OUT (host-to-device) transaction.
	IN	1001	all	host	all	Device and endpoint address for IN (device-to-host) transaction.
	SOF	0101	Start-of-Frame	host	all	Start-of-Frame marker and frame number.
	SETUP	1101	control	host	all	Device and endpoint address for Setup transaction.
Data (carries data or status code)	DATA0	0011	all	host, device	all	Data toggle, data PID sequencing
	DATA1	1011	all	host, device	all	Data toggle, data PID sequencing
	DATA2	0111	isoch.	host, device	high	Data PID sequencing
	MDATA	1111	isoch., split transac-tions	host, device	high	Data PID sequencing
Handshake (carries status code)	ACK	0010	all	host, device	all	Receiver accepts error-free data packet.
	NAK	1010	control, bulk, interrupt	device	all	Receiver can’t accept data or sender can’t send data or has no data to transmit.
	STALL	1110	control, bulk, interrupt	device	all	A control request isn’t sup-ported or the endpoint is halted.
	NYET	0110	control Write, bulk OUT, split transac-tions	device	high	Device accepts error-free data packet but isn’t yet ready for another or a hub doesn’t yet have complete-split data.

Figura 29. Tipos de paquete y su identificador (USB Complete Third Edition)

2.10.3. Tipos de transferencia:

El USB está diseñado para manejar varios tipos de periféricos con variaciones de requisitos de tasa de transferencia, tiempo de respuesta y corrección de errores. Se establecen, por tanto, cuatro tipos de transferencia según las necesidades del dispositivo.

Se comentarán las 3 primeras y se desarrollará en el siguiente apartado la última por ser motivo de estudio

- **Control:** Definido por la especificación USB. Habilita al anfitrión a leer la información sobre los requisitos del fabricante del dispositivo, asignarle una dirección y seleccionar la configuración y otros ajustes.
- **Mayoritario ("Bulk"):** Se utiliza para cuando el índice de transferencia no es crítico. Como impresoras, o unidades de disco externo. En la que una alta velocidad es requerida, pero el dato podría esperar si fuese necesario.
- **Isócrona ("Isochronous"):** Utilizada para aquellos dispositivos en los que hay que garantizar una tasa de transferencia determinada, por ejemplo cámaras. En este caso no está permitida la retransmisión en caso de error.
-

Transfer Type	Control	Bulk	Interrupt	Isochronous
Typical Use	Identification and configuration	Printer, scanner, drive	Mouse, keyboard	Streaming audio, video
Required?	yes	no	no	no
Low speed allowed?	yes	no	yes	no
Data bytes/millisecond per transfer, maximum possible per pipe (high speed).*	15,872 (thirty-one 64-byte transactions/ microframe)	53,248 (thirteen 512-byte transactions/ microframe)	24,576 (three 1024-byte transactions/ microframe)	24,576 (three 1024-byte transactions/ microframe)
Data bytes/millisecond per transfer, maximum possible per pipe (full speed).*	832 (thirteen 64-byte transactions/ frame)	1216 (nineteen 64-byte transactions/ frame)	64 (one 64-byte transaction/ frame)	1023 (one 1023-byte transaction/ frame)
Data bytes/millisecond per transfer, maximum possible per pipe (low speed).*	24 (three 8-byte transactions)	not allowed	0.8 (8 bytes per 10 milliseconds)	not allowed
Direction of data flow	IN and OUT	IN or OUT	IN or OUT (USB 1.0 sup-ports IN only)	IN or OUT
Reserved bandwidth for all transfers of the type (percent)	10 at low/full speed, 20 at high speed (minimum)	none	90 at low/full speed, 80 at high speed (isochronous & interrupt combined, maximum)	
Error correction?	yes	yes	yes	no
Message or Stream data?	message	stream	stream	stream
Guaranteed delivery rate?	no	no	no	yes
Guaranteed latency (maximum time between transfers)?	no	no	yes	yes
*Assumes transfers use maximum packet size.				

Figura 30. Tabla de diferencias de transferencias (USB Complete Third Edition)

2.10.4. Transferencia por Interrupciones:

Utilizada cuando resulta útil enviar datos en un intervalo específico de tiempo. Como por ejemplo en teclados, ratones, mandos de videojuegos y para los notificadores de estado de los hubs. En estos dispositivos, el usuario no debe percibir un retraso significativo desde que comete la acción hasta que se refleja. Es sin duda el tipo de transferencia más utilizado por los "HID" y, por tanto, el estudiado en este proyecto

Pese a su nombre, funciona al igual que el resto de tipos de transferencia, el dispositivo no se comunica directamente con el anfitrión, sino que espera hasta que éste se comunica con él. Sin embargo, el anfitrión garantiza la solicitud o envío de información a intervalos con mínimo retraso.

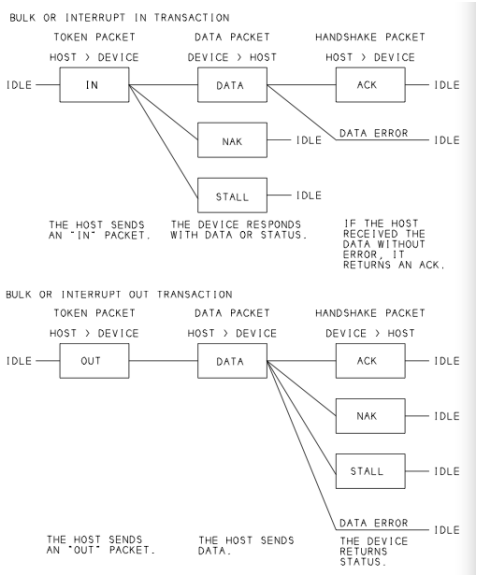

Figura 31. Esquema de transferencia Interrupt (USB Complete Third Edition)

Según la velocidad del USB, el descriptor del chip USB especifica la máxima latencia (o retraso) con el que el anfitrión puede comunicarse con el periférico, y pese a que la velocidad no está garantizada de ningún modo, también se limita el número máximo de paquetes:

- Baja velocidad ("low-speed"): 10-255 ms, 19 Bytes/paquete
- Velocidad completa ("full-speed"): 1-255 ms, 13Bytes/paquete
- Alta velocidad ("high-speed"): 125 us - 4 s (en incrementos de 125us), 55 Bytes/paquete.

Nótese que se establece un límite en tanto en cuanto una transmisión de alta velocidad de interrupciones, que combinada con una isócrona, no puede usar más del 80% del intervalo de comunicación

El tratamiento de errores es similar a otros modos, ya que el anfitrión reintenta hasta tres veces si no recibe el aceptado ("handshake"), en cambio, reintenta sin límite si el dispositivo no puede enviar o aceptar más transferencias ("NAK").

2.11. Estados USB

2.11.1. Enumeración

Como se ha comentado antes, para que una aplicación se pueda comunicar con un dispositivo, el anfitrión debe recabar toda la información del periférico y así, asignarle un controlador ("driver"). Y aunque no sea estrictamente necesario conocer todos los pasos para poder programar el firmware, sí que es interesante a título informativo.

El proceso de enumeración sigue este orden, aunque suponerlo en la programación del firmware puede llevar a error, pues en función de la aplicación, puede haber variaciones:

1. **El usuario conecta un dispositivo a un puerto USB.** El hub suministrará potencia al dispositivo, que pasará a estar alimentado.
2. **El hub detecta el dispositivo**. El hub monitoriza las variaciones de tensión en las líneas de datos que se producen. El hub tiene una resistencia "pull-down" de 15-25Kohms, y el dispositivo una de "pull-up" de 900-1500ohms a D+, si es un dispositivo de velocidad completa, o D-, si es de baja velocidad.
3. **El anfitrión recaba información del nuevo dispositivo**. El hub utilizará su interrupción "punto final" para hacer llegar al anfitrión de que ha habido un evento. Entonces el anfitrión enviará la orden "Get_Port_Status" al hub, que responderá indicando que hay un nuevo dispositivo conectado.
4. **El hub detecta la velocidad del dispositivo**. Antes del reset el periférico, el hub mide en qué línea de datos se ha producido la variación de tensión y determina la velocidad. Si fuese un dispositivo de alta velocidad se determinará en fases siguientes
5. **El hub resetea el dispositivo**. Cuando el anfitrión es informado del nuevo dispositivo, su controlador envía al hub una solicitud de "Set_Port_Feature" para que se reinicie el dispositivo. El hub pondrá D- y D+ a bajo nivel durante al menos 10 ms sólo en ese puerto.
6. **El anfitrión detecta si el dispositivo es de velocidad completa o alta velocidad.** Durante el reset, si el chip del dispositivo admite alta velocidad enviará una serie de comandos "Chirp J" y "Chirp K" alternados.
7. **El hub establece una vía de señales entre el dispositivo y el bus.** El anfitrión sigue enviando solicitudes de "Get_Port_Status" hasta que el dispositivo haya salido del estado reset o se agote el tiempo de reset. Cuando el hub retira el reset del dispositivo, los registros de éste se han reseteado y está listo para responder a las transferencias de control del "punto final" 0. El dispositivo puede ahora comunicarse con el anfitrión en con la dirección por defecto 00h y puede tomar hasta 100mA del bus.
8. **El anfitrión envía "Get_Descriptor"** para solicitar al dispositivo el tamaño máximo del paquete de la tubería por defecto. Lo envía a la dirección del dispositivo 0 y "punto final" 0. Si hubiese varios dispositivos conectados al mismo tiempo, sólo uno respondería a esa dirección. En este punto empieza la fase "Status"

9. **El anfitrión asigna una dirección.** Envía una única dirección al dispositivo mediante la orden "Set_Address". En este punto acaba la fase de "Status" y empieza la fase "Dirección", implementando esta nueva dirección en su sistema. Esta dirección permanecerá hasta que el dispositivo se apague o se desconecte del bus.
10. **El anfitrión recaba información sobre el dispositivo**. El anfitrión envía la solicitud "Get_Descriptor" a la nueva dirección del dispositivo, a su "punto final" 0 y lee el descriptor completo del dispositivo. El anfitrión continúa recabando información solicitando uno o más descriptores de configuración especificados en el descriptor del dispositivo, y así sucesivamente con los descriptores de configuración subordinados definidos en sendos descriptores de configuración.
11. **El anfitrión asigna y carga un controlador software ("driver") para el dispositivo**. A excepción de los dispositivos combinados. Varía en función del sistema operativo, sin embargo, tras esta selección, el sistema operativo puede comunicarse directamente utilizando las clases del driver.
12. **El controlador software selecciona la configuración**. Tras escoger el controlador, éste solicita la configuración a través de la orden Set_Configuration. Este paso depende totalmente de la programación escogida y el dispositivo quedará listo para utilizarse.

2.11.2. Estados adicionales:

Adjunto. Sucede cuando el hub no suministra potencia a un dispositivo a través de las líneas de tensión. Puede ser debido si el hub ha detectado una demanda demasiado alta de corriente o si el anfitrión ha solicitado al hub que retire la alimentación del puerto. Se tratará al dispositivo como si no hubiese sido conectado

Suspendido. Un dispositivo entra en este estado cuando, tras cómo mínimo 3 milisegundos, no detecta actividad en el bus. En este estado deberá limitar su consumo de corriente del bus y deberá estar preparado para entrar en este estado aún sin haber sido configurado.

2.11.3. Retirada del dispositivo:

El anfitrión es informado por el hub de un evento al desabilitar el puerto del dispositivo. El anfitrión entonces solicita la información del hub a través de la orden "Get_Port_Status" para obtener la información específica. Ahora el sistema operativo deberá eliminar el controlador del dispositivo del gestor.

2.12. Descriptores USB.

Los descriptores USB son bloques estructurados de información que habilitan al anfitrión para recabar información acerca del dispositivo. El dispositivo debe almacenar la información en los descriptores y responder a las solicitudes de descriptores del anfitrión.

Existen cinco descriptores estándar:

1. **Descriptor de dispositivo**. Único por dispositivo, describe información general que se aplica al dispositivo y a todas sus posibles configuraciones.
2. **Descriptor de configuración**. Describe información sobre una configuración específica, puede tener varios descriptores. Cada configuración tiene una o más interfaces y cada interfaz puede tener 0 o más "puntos finales" (representación lógica de una parte de hardware del dispositivo que interactúa con el sistema USB). Los endpoints ("puntos finales") no se comparten entre interfaces dentro de una misma configuración, a menos que el "punto final" se utilice con configuraciones alternativas de la misma interfaz.
3. **Descriptor de interfaz**. Describe un interfaz dentro de una configuración. Un interfaz puede tener varias configuraciones permitiendo a los endpoints ("puntos finales") variar después de que el dispositivo haya sido configurado. La configuración por defecto es la cero.
4. **Descriptor de endpoint ("punto final")**. Definen los requisitos de transmisión de cada dispositivo. El "punto final" cero se usa para control y no tiene descriptor.
5. **Descriptor de cadenas**. Son opcionales y dan información adicional: el nombre del fabricante, del dispositivo o el número de serie. Son legible en formato Unicode.

En la página http://www.beyondlogic.org/usbnutshell/usb5.htms se puede encontrar una descripción detallada.

2.13. Hardware USB

Para crear un dispositivo USB existen generalmente dos vías. O bien utilizar un circuito integrado de aplicación específica ("ASIC") o bien utilizar un microprocesador adecuado para trabajar con USB, que es en esencia, un microcontrolador con un ASIC-USB embebido. Existe también una tercera forma que sería mediante el uso de FPGAs, dada su alta velocidad, pero esta vía es bastante específica dada su complejidad en comparación con las otras implementaciones.

El controlador (igualado al ASIC, de manera conceptual), constará de:

- **Transceiver:** Es el interface hardware entre el conector USB y el circuito que controla la comunicación USB.
- **El motor de interfaz serie ("SIE").** Encargado de enviar y recibir datos de las transacciones y comprobar errores a través del bit de comprobación de errores ("CRC"). No interpreta estos datos.
- **Almacenes temporales ("Buffers").** El controlador USB utiliza estos almacenes temporales para almacenar datos recibidos y los datos que están listos para ser enviados.
- **Registros de configuración, status y control.** En estos registros fijos se almacena todo tipo de información acerca del dispositivo que representa y ocupan posiciones determinadas para el fácil acceso por parte del anfitrión.
- **Reloj.** Para las comunicaciones usb es necesario un reloj. En aplicaciones de baja velocidad, éste puede ser integrado.

2.14. HID-USB

2.14.1. Introducción

Como se comentó en el apartado de los tipos de transferencia, los dispositivos de interfaz humana especifican un tipo de trasferencia por interrupciones. Sus características son las que siguen:

- Todos los datos intercambiados residen en estructuras llamadas avisos ("reports"). El anfitrión envía y recibe datos enviando solicitudes de avisos en transferencias de control o de interrupción.
- Un interfaz HID debe tener un "punto final" IN de interrupción para enviar avisos de entrada y un máximo de un "punto final" OUT de interrupción. Si se necesitan más "puntos finales" de interrupción se deberá usar un sistema de composición de dispositivos.
- El "punto final" IN de interrupción, habilita al HID a enviar información al anfitrión en intervalos impredecibles, ya que no hay forma de que el anfitrión sepa cuando se activa, o presiona, por ejemplo, una tecla de un teclado USB.
- La velocidad máxima de transmisión será de 800bytes/s para baja velocidad, 64Kbytes/s para velocidad completa y de 24Mbytes/s para alta velocidad, pero esta velocidad no garantizada, sólo representa su máximo valor alcanzable.

2.14.2. Puntos finales.

Los puntos finales de un dispositivo HID deberá tener esta estructura

Tipo de transferencia	Origen de los datos	Datos usuales	¿Necesita tubería?
Control	Dispositivo (IN)	Datos que no dependen de un tiempo crítico	Si
Control	Anfitrión (OUT)	Datos que no dependen de un tiempo crítico o cualquier dato si no hay una tubería OUT de interrupción	Si
Interrupt	Dispositivo (IN)	Periódicos o de baja latencia	SI
Interrupt	Anfitrión (OUT)	Periódicos o de baja latencia	No

2.14.3. Avisos.

El requisito de un "punto final" de interrupción IN, sugiere que todos los HID deben tener, como mínimo, un aviso de entrada definido en el descriptor de avisos del HID. Los avisos de salida y de características son opcionales.

2.14.4. Transferencias de control.

Las especificaciones HID definen seis solicitaciones específicas de clase.

- A través de **"Set_Report"** el anfitrión envía avisos de configuración al dispositivo y a través de **"Get_Report"** los recibe.
- **"Set_Idle"** y "**Get_Idle"** son usados por el anfitrión para establecer la inactividad del dispositivo en función de si ha variado o no su estado desde las últimas consultas**.**
- **"Set_Protocol"** y **"Get_Protocol"** demandan leer o activar el valor de protocolo a utilizar para la comunicación en función del driver del sistema operativo.

2.14.5. Transferencias de Interrupción.

Los "puntos finales" de interrupción proveen una manera alternativa de intercambiar datos, especialmente cuando el destinatario debe obtener los datos periódicamente o con determinada celeridad. Las transferencias de control pueden ser demoradas si el bus está muy ocupado, sin embargo, el ancho de banda para la transferencia por interrupción debe estar garantizado cuando se configura el dispositivo.

2.14.6. Requisitos Firmware.

El firmware del dispositivo debe también cumplir ciertos requisitos de clase.

- **El descriptor del dispositivo debe incluir un descriptor del interfaz que especifique el tipo de HID, un descriptor del HID y un descriptor del "punto final" IN de interrupción.** La descripción del punto final OUT de interrupción es opcional. El firmware debe contener también un descriptor de avisos que contiene la información acerca de los contenidos de los avisos del HID.
- **Un HID puede soportar uno o más avisos**. El descriptor de avisos especifica el tamaño y los contenidos de los datos de los avisos del dispositivo y deben incluir información sobre como el destinatario de los datos debe usarlos. Los valores en el descriptor definen cada aviso como avisos de entradas, salidas o característica, siendo entrada y salida desde el punto de vista del anfitrión y los avisos de característica pueden viajar en cualquier dirección.

2.14.7. Identificando un dispositivo como HID.

Como cualquier dispositivo USB, un descriptor HID le dice al anfitrión qué necesita saber para comunicarse con el dispositivo. El anfitrión recaba información sobre el HID durante la fase de enumeración enviando la solicitud de "Get_Descriptor", para la configuración contenida en el interfaz del HID.

El descriptor de configuración del interfaz identifica el interfaz como un tipo HID. El descriptor del tipo de HID especifica el número de descriptores de avisos soportados por la interfaz. Durante la enumeración, el controlador ("driver") del HID demanda el descriptor de aviso y cualquier descriptor físico.

2.14.8. Transfiriendo datos

Cuando la enumeración se completa, el anfitrión ha identificado el interfaz del dispositivo como un HID y ha establecido tuberías con los puntos finales del interfaz y aprendido qué formatos de avisos usar para la comunicación.

El anfitrión puede entonces demandar avisos usando transferencias de interrupción IN y/o transferencias de control con la solicitud **"Get_Report".** El dispositivo también tiene la opción de soportar avisos de interrupción OUT y/o transferencias de control con solicitudes tipo **"Set_Report"**.

2.15. LUFA

LUFA[12] ("Lightweight USB Framework for AVRs") es un proyecto de código abierto creado y mantenido por Dean Camera que provee del stack completo USB para microcontroladores Atmel de la serie AT90USBxxxx y ATMEGAxxUx. Está licenciado bajo la permisiva licencia MIT y está escrito para el compilador libre AVR_GCC.

Figura 32. Logotipo de LUFA

Se puede encontrar, incluido en la librería fuente, varias aplicaciones de demostración que muestran el uso de la librería. Actualmente las librerías incluyen, demostraciones de bootloaders para distintos sistemas y las siguientes aplicaciones de demostración:

Dispositivos:

- Entrada de audio
- Salida de audio
- Serie doble virtualización
- HID Genérico
- Joystick
- Teclado
- Teclado/ratón
- Almacenamiento masivo
- Almacenamiento masivo/teclado
- MIDI
- Ratón
- Impresora
- RNDIS Ethernet
- Serie virtual
- Serie virtual/ratón

Anfitriones

- HID genérico
- Joystick
- Teclado
- Almacenamiento masivo
- MIDI
- Raton
- Impresora
- RNDIS Ethernet
- Serie virtual

Es un código programado de forma modular, de tal forma que el mismo núcleo del USB puede ser incluido en cualquiera de los códigos demo que adjunta. Además está preparado para detectar, a modo de driver, las placas con sus elementos hardware en la que se incluye el micro.

Este stack será el utilizado en este proyecto, más específicamente, se trabajará a partir de la aplicación de demostración del dispositivo Joystick. De esta forma sólo se necesitará trabajar la demo del Joystick y el driver de la placa, dejando intacto el núcleo de la pila USB.

3. Desarrollo del Proyecto

3.1. Visión general

3.1.1. Objetivos

Se quiere desarrollar un dispositivo de interfaz humana (HID) USB tipo Joystick (tanto el hardware como el software) de 2 botones, con la salvedad de que no utilizará palanca, en vez de ello, enviará la inclinación detectada por el acelerómetro (figura 33).

El dispositivo se debe identificar como dispositivo USB, y en el bucle de aviso de interrupción USB, debe enviar la información correspondiente a la inclinación del acelerómetro y las pulsaciones de los botones. El desarrollo se realizará sobre un microcontrolador AVR de ATMEL.

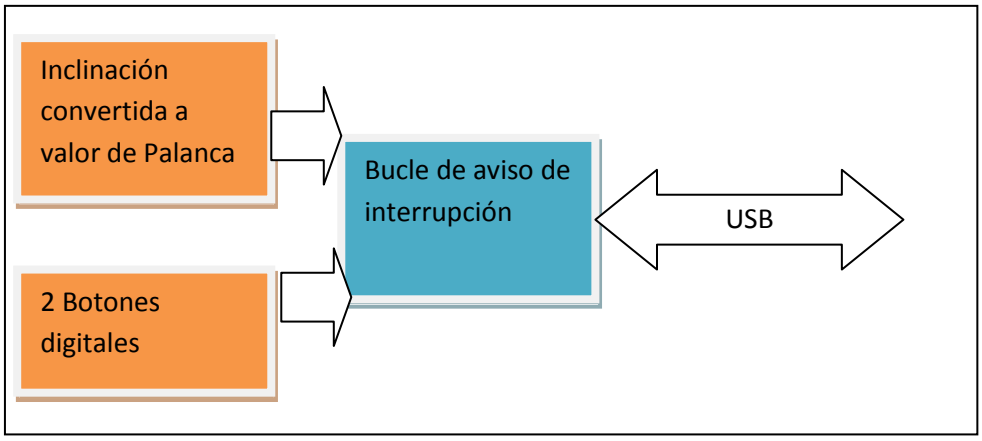

Figura 33. Esquema de objetivos básicos

Puesto que, para las pruebas durante el desarrollo, se han implementado un array de ocho LEDs, una bocina controlada por un PWM de 16 bits, y dos potenciómetros a ser leídos por el ADC del microcontrolador, los objetivos se extienden a:

- La pulsación del botón 1, generará un tono en DO 3ª octava
- La pulsación del botón2, generará un tono en RE 3ª octava
- La pulsación del botón 1 y 2, generará un tono en Mi 3ª octava
- La variación del potenciómetro 1, variará de forma lineal el número de LEDs encendidos del array
- La variación del potenciómetro 2, variará el volumen del altavoz.

Una vista esquemática de las entradas, salidas y componentes del proyecto se muestra en la figura 34. En ella se pueden ver los elementos hardware a instalar en la placa ("perfboard") y el puerto que usan del microcontrolador, representado por la flecha.

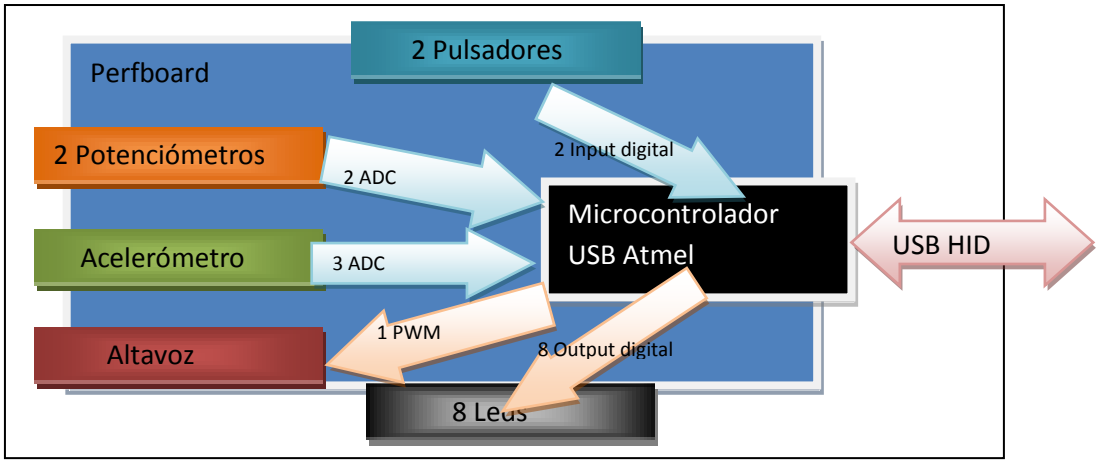

Figura 34. Visión global del proyecto

En el apartado 3.2 se detalla el diseño de cada uno de los elementos de que consta el proyecto.

3.1.2. Placa para pruebas del microcontrolador USB

El gran problema que presentan los microcontroladores AVR es que no presentan modelos USB con empaquetado DIP (el de mayor tamaño), por lo que si se quiere realizar proyectos no profesionales, para pruebas o para este proyecto, por ejemplo, se debe recurrir a adaptadores o placas que creen un interfaz mecánico de mayor tamaño con el que poder trabajar sobre una protoboard o sobre una perfboard.

Dada la filosofía de este proyecto, el más acorde hubiese sido el sistema Micropendous, de arquitectura abierta (open hardware), con una comunidad detrás que desarrolla aplicaciones y programas para su uso y prueba. Pero al ser un producto escasamente comercial, resulta muy complicado adquirir uno. Por otro lado, fabricarlo tampoco era una opción válida dentro de los objetivos y requisitos del proyecto.

Figura 35. Modelos de micropendous

La opción escogida fue el Teensy2++, de pjrc.com [11]. Al igual que el Micropendous, es, en esencia, un zócalo con un conector USB, un botón de reset y un LED para un microcontrolador Atmel AVR AT90USB1286.

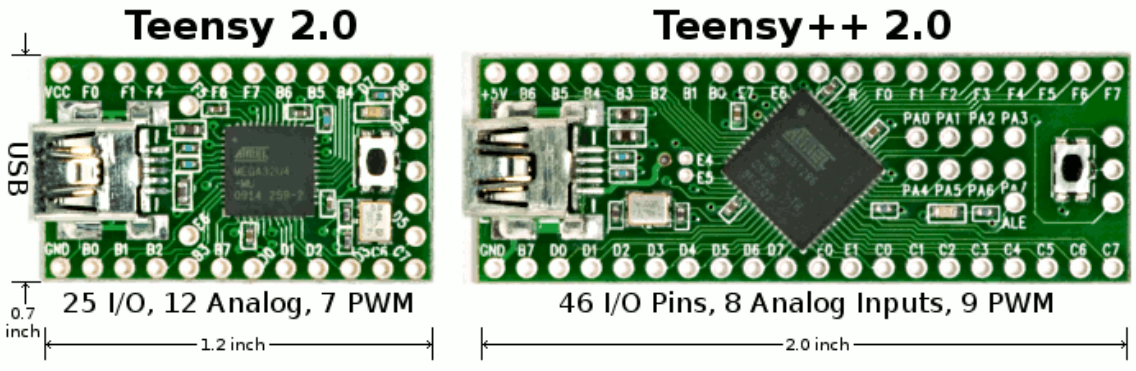

Figura 36. Modelos de Teensy 2.0

El creador ha dispuesto en su página web de códigos de ejemplo para usar en los Teensy y herramientas multiplataforma para programar el micro a través del bootloader preconfigurado con tan sólo un cable USB.

3.1.3. Diseño Hardware

El esquema electrónico del proyecto completo, que se detallará en sucesivos apartados, es el que se muestra en la figura 37. En él se puede apreciar la conexión física de los distintos elementos.

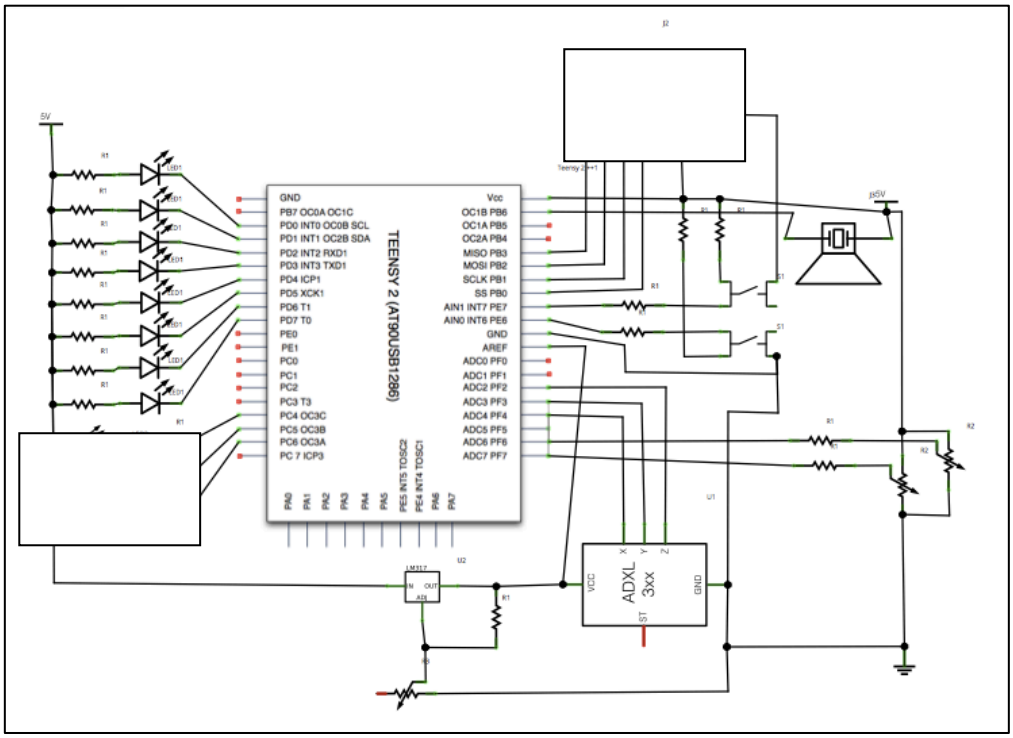

Figura 37. Esquema del circuito completo

Salvo los elementos dentro de los cuadros blancos, que fueron instalados para pruebas y futuras aplicaciones del prototipo, la figura 37 representa el esquema eléctrico, en la figura 38 se pueden ver imágenes de cómo quedaría en una perfboard.

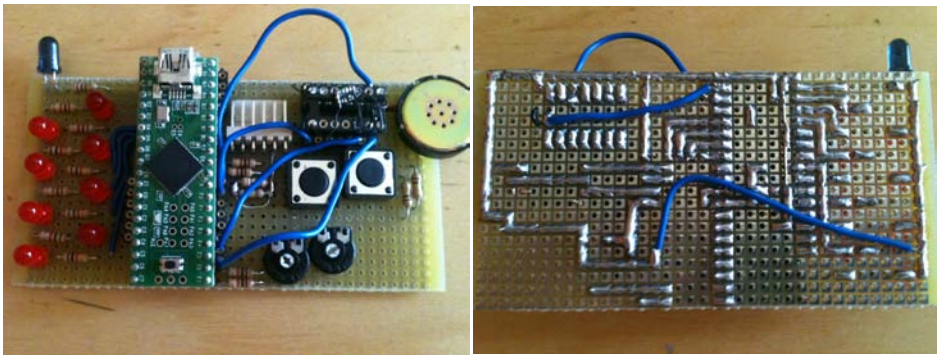

Figura 38. Fotografía del prototipo

Como se comentará en el apartado 3.2.5, hubo que adaptar el encapsulado a un zócalo de 14 pines para poder trabajar con él, como se muestra en la figura 39.

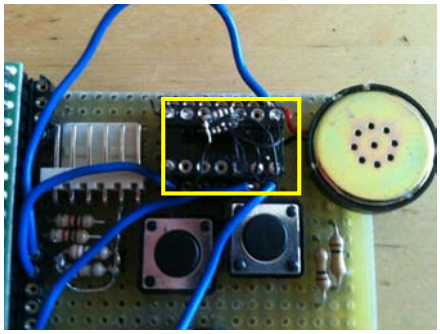

Figura 39. Detalle de la adaptación del acelerómetro

Por problemas de espacio en la perfboard elegida, el circuito convertidor de tensión para alimentar el acelerómetro se ha insertado debajo del Teensy 2++, en el hueco dejado por el zócalo (figura 40). En el detalle resaltado en amarillo se puede apreciar el potenciómetro elegido para poder ser calibrado con la placa Teensy instalada. Además en el detalle en azul se puede ver cómo se ha dispuesto un jumper que cortocircuita la salida del elemento regulador con la entrada de referencia analógica.

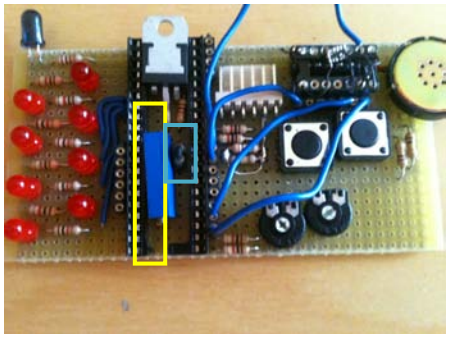

Figura 40. Detalle de elemento regulador de tensión.

3.1.4. Diseño Software

Para simplificar y reutilizar el código creado para el proyecto se han creado las librerías siguiendo un esquema de capas o estratificado como se esquematiza en la figura 41.

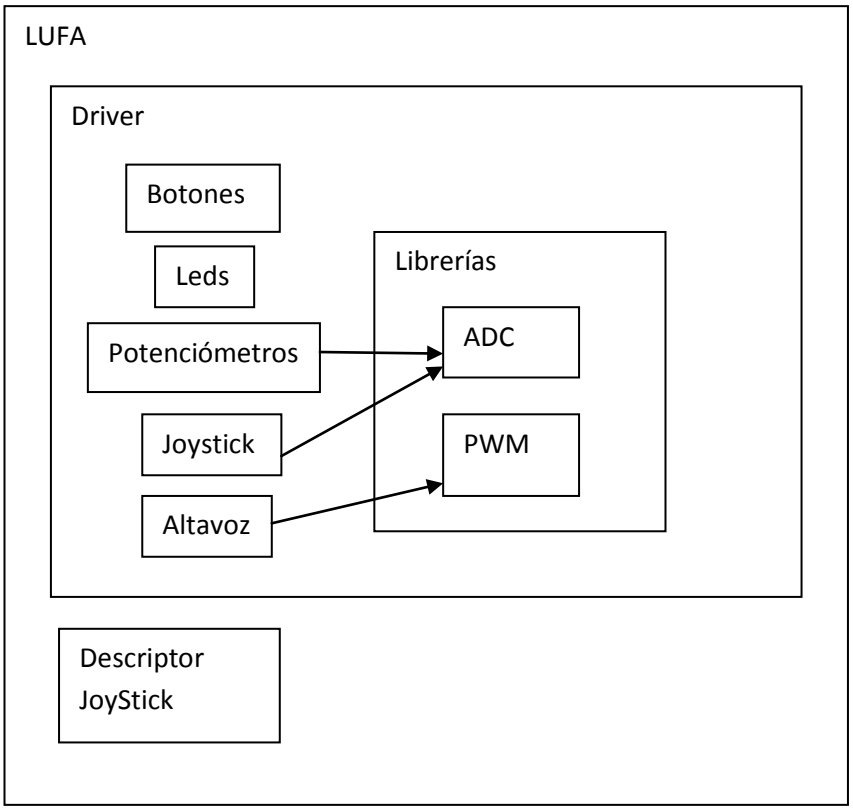

Figura 41. Esquema de dependencias software

Según esta figura, como se detallará en el apartado de descripción del interfaz, el código se ha dispuesto a partir del stack USB, LUFA, que engloba todo el proyecto. Dentro y a modo de driver, para que el núcleo pueda actuar sobre el hardware utilizado, se han incluido los distintos elementos a modo de librería. Además, contemplando la reutilización eventual de estas librerías, se ha procurado estratificar en la medida de lo posible, de ahí que se hayan escindido parte de los módulos, como el ADC y el PWM, para que puedan ser utilizados por otros elementos (según la figura, relacionados con flechas).

Cabe mencionar la inclusión de "Descriptor Joystick", pues como se detallará en el apartado de configuración del dispositivo y de los descriptores, es necesario modificar este archivo para poder utilizar el hardware.

3.2. Estudio detallado de los elementos

Todos los elementos, bien sean físicos o conceptos que utilicen varios elementos físicos, de describirán con la misma estructura:

- **Introducción:** Introducción general del elemento
- **Funcionamiento eléctrico:** Cómo se comporta al serle aplicado una corriente
- **Uso en un microcontrolador:** Prueba de funcionamiento del elemento por separado y regido por un microcontrolador.
- **Uso específico dentro del proyecto:** Repertorio de funciones creadas para su control en el proyecto

3.2.1. LEDs.

a. Introducción:

Un "Light Emitter Diode" (diodo emisor de luz), es un diodo que, en función de la corriente que circule desde el ánodo hasta el cátodo, emitirá luz con una potencia proporcional.

Tras su acondicionamiento eléctrico, dada su sencillez, resulta el elemento ideal para pruebas de salida digital en la programación de microcontroladores.

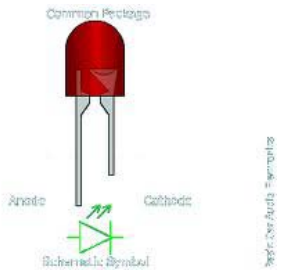

Figura 42. Esquema de un LED

b. Funcionamiento eléctrico

Existen varios tipos de LEDs, en función del color (o longitud de onda) que emitan, con rangos que van desde el azul hasta el infrarrojo. Aunque en todos los casos el funcionamiento es análogo. En la figura 43 se puede encontrar una referencia del color en función de la longitud de onda

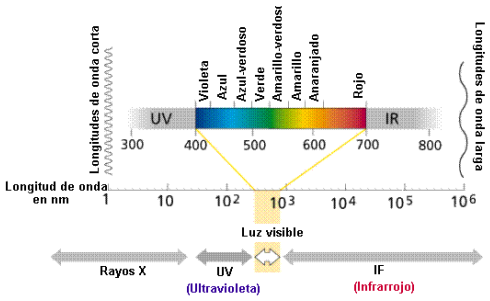

Figura 43. Espectro de luz

Para acondicionar eléctricamente un LED, se debe conocer la tensión umbral y la curva de Intensidad vs Potencia lumínica. En el caso de un LED rojo estándar, por ejemplo, según la figura 44, se puede observar que la máxima potencia lumínica se obtiene haciendo conducir por este elemento una corriente de 30mA:

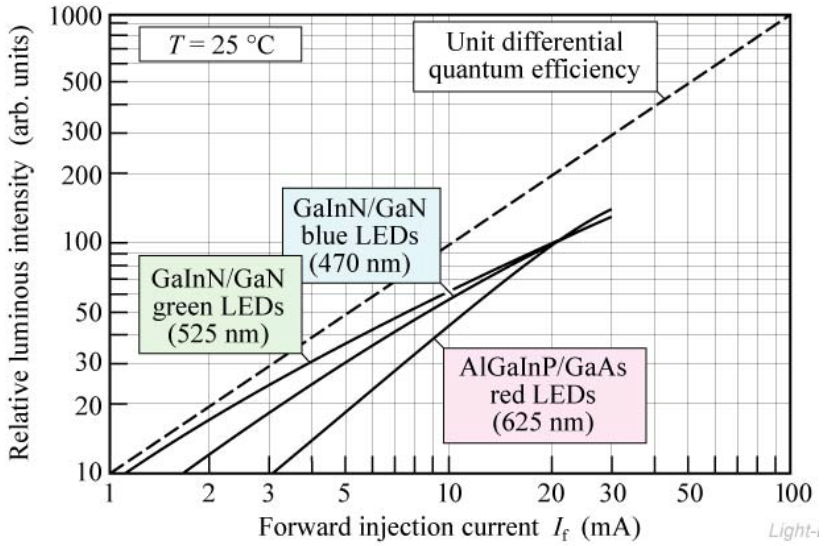

Figura 44. Curvas características Intensidad vs luminosidad relativa

Por tanto si se va a trabajar con una tensión de alimentación de 5V, se conoce que la tensión umbral equivale a 1,8V y se quiere hacer correr una corriente máxima de 0,03A, el diodo debe estar en serie (figura 45), con una resistencia mínima siguiendo el siguiente cálculo:

$$R_{min}=(V_{cc}-V_{umbral})/I_{max} = (5V-1,8V)/0,03A= 106\ \Omega$$

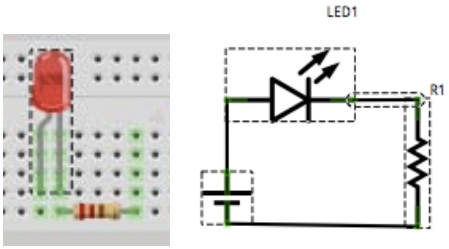

Figura 45. Conexión en continua de un LED

Esta resistencia calculada sería la que permitiese la máxima potencia de luz de salida, sin embargo, puesto que en el proyecto total existen más elementos y se quiere que el consumo sea razonablemente bajo, la resistencia elegida será casi diez veces superior (1KOhm), para conseguir una potencia visible con el mínimo costo energético.

Es interesante señalar que si se desea accionar varios LEDs independientemente, cada uno deberá llevar su propia resistencia.

c. Uso en un microcontrolador.

Existen dos formas de accionar un LED en función del acondicionamiento, bien por nivel alto o por nivel bajo. Es decir, el LED puede ser alimentado por la salida digital del microcontrolador, y descargarse en la masa, o puede estar conectado a la alimentación general (5V para este estudio) y para descargarse utilizará la salida digital cuando toma el valor 0.

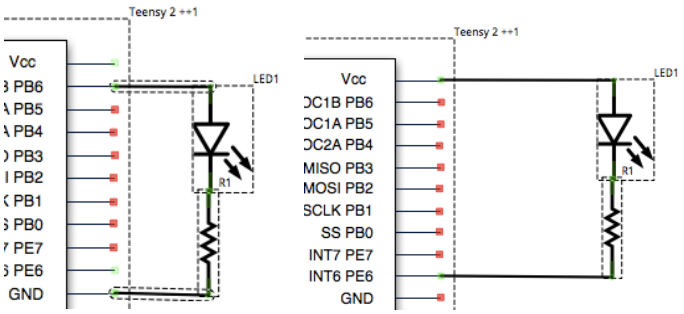

Figura 46. Conexión de un LED en un microcontrolador

Aunque ambas formas son perfectamente válidas, se recomienda el uso de la segunda forma, pues en el caso de este proyecto, por ejemplo, toman la alimentación del mismo punto que el microprocesador, y no es éste el que los tiene que alimentar, con el consecuente aumento de temperatura. Véase que la decisión de uno u otro acondicionamiento determina el valor de salida a utilizar para accionar el LED.

Desde el punto de vista de la programación de un microcontrolador, para imponer una salida digital en un puerto, se debe modificar el registro PORTx, siendo x el puerto elegido, en este caso el B.

El ejemplo que se suele utilizar para la primera programación suele ser encender y apagar un LED. En el caso de que sea activo por nivel bajo y conectado al pin 6 del puerto B, el ejemplo, que encendería y apagaría un led cada 100ms sería el que se muestra en la Figura 47.:

```
#include<avr/io.h>          //contiene las definiciones de los registros del microcontrolador
#include<util/delay.h>//Contiene la función "_delay_ms()"
voidmain (void)
{
        DDRB = 0b01000000; // Al poner a 1 el PIN6 del latch del puerto B, se //
configura como salida
        while (1)                    // bucle infinito
        {
                PORTB = 0b01000000;              // A 1 se apaga el LED, entonces LED
OFF
                _delay_ms(100);          // espera 100ms
                PORTB = 0b00000000;              // LED ON
                _delay_ms(100);          // espera 100ms
        }
}
```

Figura 47. Ejemplo de programa para controlar un led

d. LEDs del Proyecto

Para este proyecto se ha dispuesto de un array de 8 LEDs que ocupan todo el puerto D del microcontrolador, activos a nivel bajo y con una resistencia en serie de 1KOhm.

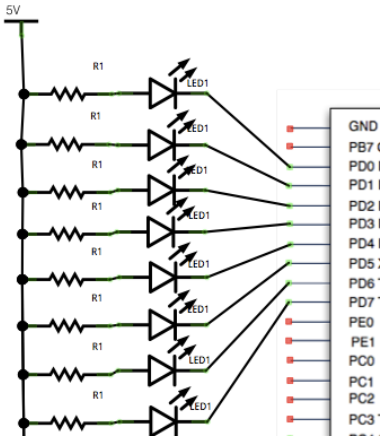

Figura 48. Conexión de array de LEDs en micro del proyecto

```
/Extracto de LEDs.h

//=====================================================================

// Configuración

//=====================================================================

#define LEDS_RED_ARRAY_CONFIG              DDRD

#define LEDS_RED_ARRAY                     PORTD

//=====================================================================

// Funciones

//=====================================================================

void leds_array_scan();

void leds_meter (uint8_t);// Enciende más o menos leds en función del valor de entrada 0-255

void leds_on(uint8_t);          //Enciende los leds según mascara de entrada de 8bits

void leds_off(uint8_t);//Apaga los leds según máscara de entrada 8bits

void leds_change(uint8_t);// Alterna encendido y apagado de leds según máscara
```

Figura 49. Resumen de funciones de control de LEDs

Por tanto, las funciones desarrolladas son:

Scan de array de LEDs

- Función**:** leds_array_scan()
- Prototipo: void leds_array_scan();
- Descripción: Enciende secuencialmente cada LED,los activa en orden de arriba a abajo y vuelve.
- Observaciones: Es una función útil para incluir en un estado y tener un indicador visual de que se ha llegado a él

Medidor LED

- Función**:** leds_meter(valor)
- Prototipo: void leds_meter(uint8_t);
- Descripción: A partir de esta función se encienden de manera proporcional los 8 LEDs en función de un valor de entrada de 8 bits. Es decir, para un "valor" =0, no encenderá ningún LED, para un "valor" =255 encenderá los 8 LEDs
- Observaciones: Proporciona información visual de variables de 8 bits. Útil para pruebas de medición de señales analógicas digitalizadas

Encendido de LEDs

- Función**:** leds_on(valor)
- Prototipo: void leds_on(uint8_t);
- Descripción: Tomando "valor", como un número binario de 8 bits, enciende los LEDs correspondientes del array. De tal forma que si valor es 33 (en binario 100001), encenderá el "LED0" y el "LED5"
- Observaciones: Si ya había LEDs encendidos, los respeta.

Apagado de LEDs

- Función**:** leds_off(valor)
- Prototipo: void leds_off(uint8_t);
- Descripción: Al igual que la función anterior, apaga LEDs en función del valor binario valor.
- Observaciones: Si ya había LEDs apagados, los respeta.

Apagado de LEDs

- Función**:** leds_change(valor)
- Prototipo: void leds_change(uint8_t);
- Descripción: Como las dos funciones predecesoras, el valor binario indica que LEDs debe alternar, entre encendido y apagado y viceversa.

3.2.2. Botones

a. Introducción

El término botón resulta demasiado ambiguo. En este proyecto el "botón" elegido es un pulsador normalmente abierto (figura 50).

Análogamente al LED, resulta el dispositivo más sencillo para probar las entradas digitales de un microcontrolador, pues debidamente acondicionado es capaz de imponer una tensión de 5V (1 lógico) o 0V (0 lógico) en un pin de un controlador.

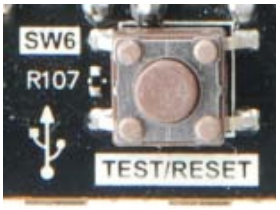

Figura 50. Imagen de un botón normalmente abierto

b. Funcionamiento eléctrico.

Un pulsador normalmente abierto es un elemento mecánico que crea un cortocircuito entre dos puntos mientras se mantiene pulsado.

Para poder imponer las tensiones deseadas, el pulsador se debe ubicar entre la tensión de alimentación y la masa, pero con una resistencia entre ambas ubicada de tal forma que no se produzca un cortocircuito en la alimentación. Por otro lado, para evitar posibles fallos a la hora de la programación que pueda deteriorar el microcontrolador, también se debe tener en cuenta en el sistema de acondicionamiento que la tensión suministrada no llegue directamente de una fuente de tensión, y pase protegida por una resistencia, sustancialmente menor que la elegida para separar la tensión de la alimentación, tal y como muestra la figura 51.

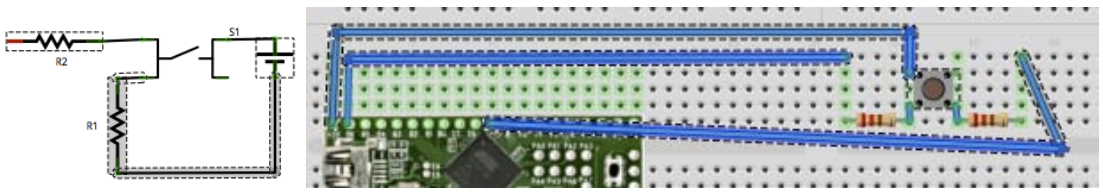

Figura 51. Conexión de un botón digital

Las resistencias elegidas son R2=1kOhms, R1=100kOhms. A simple vista se puede ver cómo sin pulsar el botón en la pata al aire de R2 se establece una tensión de 0V, y si se pulsa, se establecen 5V.

Este montaje será activo por nivel alto.

c. Uso en un microcontrolador

Para probar el uso de un pulsador y su acondicionamiento en un microcontrolador, se utilizará el esquema propuesto activo por nivel alto y las funciones creadas para el uso del array de LEDs.

A diferencia de las salidas de un puerto, como el usado para los LEDs, el registro a leer es PINx, donde x es la letra del puerto. Según el esquema, la entrada se producirá en el pin3 del puerto B.

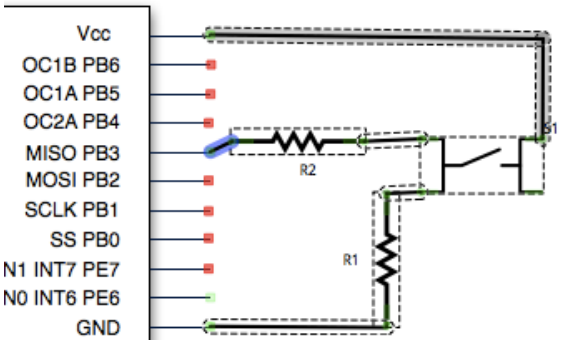

Figura 52. Conexión de un botón en un microcontrolador

Utilizando las funciones creadas para los LEDs, se puede probar el funcionamiento del botón conectado encendiendo y apagando un LED en función de la pulsación

```
#include<avr/io.h>  //contiene las definiciones de los registros del microcontrolador

#include"leds.c"

voidmain (void)

{

        DDRB = 0b00000000; // Al poner a 0 el PIN3 del latch del puerto B, se configura como entrada

        while (1)                       // bucle infinito

        {

          leds_change(PINB);//En función de la salida del puertoB, cambiará el led

        }

}
```

Figura 53. Ejemplo de prueba de un botón

d. Pulsadores del proyecto

Para el proyecto se han utilizado dos pulsadores normalmente abiertos acondicionados, cada uno con una resistencia de 1kOhm y otra de 100kOhm, y activos por nivel bajo en los pines 6-7 del puerto B.

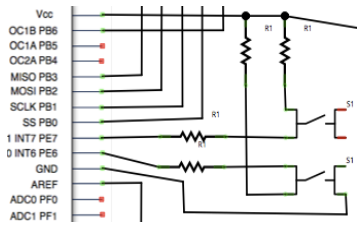

Figura 54. Esquema de conexión de dos botones al microcontrolador

```
// fragmento de buttons.h

//=====================================================================

// Config

//=====================================================================

#define BUTTONS_CONFIG  DDRE &= ~

#define BUTTONS                         PINE&

#define B1                                      (1<<6)

#define B2                                      (1<<7)

//=====================================================================

// Funicones

//=====================================================================

uint8_t buttons_adquire();      //devuelve una máscara de 8bits con botones pulsados (ordenados)
```

Figura 55. Fragmento de código relacionado con los botones

La única función perteneciente a botones del proyecto es:

Lectura de pulsaciones de botón:

- Función**:** buttons_adquire()
- Prototipo: uint8_t buttons_adquire()
- Descripción: Devuelve un valor de 8bits, que entendido como binario, cada 1 implica una posición de botón pulsado. Por ejemplo si sólo se pulsase "boton1", el valor devuelto sería 2 (0b000010)
- Observaciones: Esta función se ha diseñado para recabar información simultáneamente de todos los botones. Da información de los botones pulsados y por omisión de los no-pulsados en el momento de hacer la consulta.

3.2.3. Convertidor Analógico Digital.

a. Introducción

Un convertidor analógico digital o ADC (Analog to Digital Converter) es un elemento tal que es capaz de dar un valor digital a partir de una tensión analógica.

Dado que vivimos en un mundo analógico, es uno de los elementos más útiles para que un microcontrolador pueda captar el entorno real en el que se encuentre. Es decir, por ejemplo un termómetro digital ofrece una tensión proporcional a la temperatura que está midiendo, y la única forma que esa información puede ser utilizada por el microcontrolador, es a través del ADC. Su símbolo electrónico es el representado en la figura 56.

ELECTRICAL SYMBOL FOR ANALOG TO DIGITAL CONVERTER (ADC)

Figura 56. Símbolo del convertidor analógico digital

b. Funcionamiento eléctrico.

Un ADC, bien sea como componente separado o integrado, tiene un funcionamiento común, que consiste en recibir una tensión de referencia que utiliza para dividir entre el número de bits que da como salida. Una vez se activa, lee la tensión de entrada, que debe de estar comprendida dentro de la tensión de referencia, y asigna un valor en función de las parcelas divididas que alcance a rellenar.

Es decir, si se tiene un ADC de 8bits para medir señales en el rango de 0-5V, se tiene que aumentará una unidad cada $5/2^8$=0,0196V. Por tanto si se pretende medir una señal de 2,65V, se tendrá una salida digital de 135.

c. Uso en un microcontrolador

Los ADC que integran los microcontroladores suelen ser accesibles a partir de un multiplexor, que multiplica las salidas. Para la tensión de referencia, el microcontrolador bien suministra la tensión de alimentación, u otra tensión interna de 2,3V, o bien provee de un pin para una tensión externa, conocida como AREF.

Los ADC pueden ser utilizados como comparadores. En vez de dar valores digitales cada vez que se les requiere, pueden ser programados para que cada vez que se alcance el nivel de tensión programado, salte una interrupción. Este caso no será motivo de estudio de esta memoria.

Para controlar un ADC, tal y como se usa para este proyecto, se necesita de la configuración de los siguientes registros:

- **ADCSRA:** Sus bits contienen la información para el inicio de la conversión, configuración del modo de empleo y para el tiempo de lectura.
- **ADCSRB:** Configura el modo de lectura y, dependiendo del modelo, tiene un bit para hacer lecturas más rápidas a costa de un mayor consumo.
- **ADMUX:** Selecciona el tipo de tensión de referencia, salida de 8 o 10bits y la entrada del multiplexor a leer.
- **DIDR0:** Un registro especial, útil cuando el pin del ADC es compartido por otra entrada digital para reducir el consumo

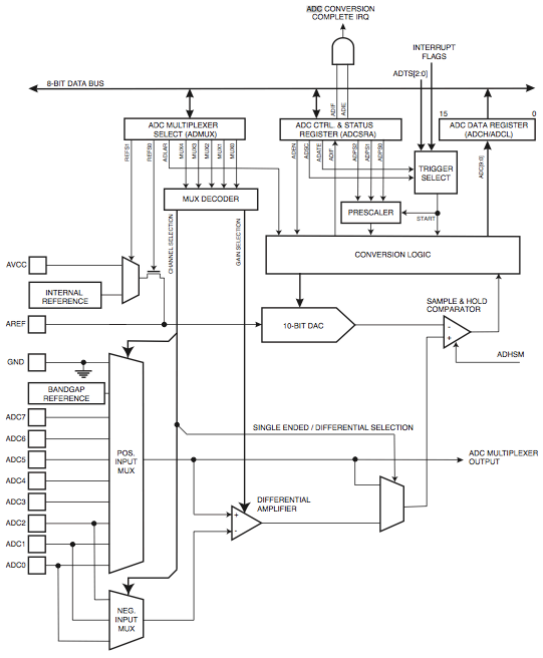

Figura 57. Esquema de un ADC dentro de un microcontrolador

Para probar el uso de un ADC con una tensión analógica se podría utilizar el código que se muestra en la figura, que como se puede ver, utiliza el medidor de LEDs, de la librería de este proyecto.

```
#include<avr/io.h>  //contiene las definiciones de los registros del microcontrolador

#include"leds.c"

#include<util/delay.h>//Contiene la función "_delay_ms()"


voidmain (void)

{

        ADCSRA = ((1<<ADEN)|(1<<ADPS2)|(1<<ADPS1));             //habilita el ADC y lo configura a máxima
velocidad

        ADCSRB = (1<<ADHSM);                    //alta velocidad

        DDRF = 0;                       // configura el puerto F como entrada

        ADMUX = ((1<<REFS0)|(1<<ADLAR)|2);// selecciona el pin 2, con referencia 5V interna y 8bits

        while (1)               //bucle infinito

        {

          ADCSRA |= (1<<ADSC);                                  // empieza la conversion

          while ((ADCSRA & (1<<ADSC))==(1<<ADSC));    //espera hasta que acaba(ADSC = 1)
        leds_meter(ADCH);               // Enciende LEDs proporcionales a la salida del adc

          _delay_ms(100);               //espera 100ms

        }

}
```

Figura 58. Ejemplo de programación de ADC

d. ADC del proyecto

En este proyecto, el ADC es usado para medir la salida de los tres ejes del acelerómetro y para dos potenciómetros. Utiliza por tanto cinco pines del puerto F.

```
//=====================================================================
// Configuración
//=====================================================================
#define ADC_PORT DDRF
//=====================================================================
// Argumentos para las funciones
//=====================================================================
//ADC_init() arguments
#define ADC_SIMPLE_FAST 0                    //High speed&clk/64
#define ADC_SIMPLE_SLOW          1           //Normal speed&clk/128
#define ADC_INT_FREE             2           //Notyet
#define ADC_OFF          3       // set allregistersto 0, stops ADC
//ADC_read(,Ref)
#define ADC_5V_REF       (1<<REFS0)                          // internal 5V reference
#define ADC_2V_REF       (1<<REFS1)|(1<<REFS0)               // internal 2,3V reference
#define ADC_EX_REF       0                                   // externalreference (AREF)
//=====================================================================
//defines internos
//=====================================================================
#define ADHSM (7) //generalmente no viene definido en las librerías avr
//=====================================================================
// Funciones
//=====================================================================
voidADC_init(uint8_t);                  //inicializa y configura el ADC
uint8_t ADC_read(uint8_t, uint8_t);     // devuelve un valor a partir de un pin del multiplexor y una referencia de tensión
(ver defines de argumenos)
```

Figura 59. Fragmento de librería de ADC utilizada en el proyecto

Las funciones de la librería son:

Inicialización del periférico ADC

- Función**:** ADC_init(valor)
- Prototipo: void ADC_init(uint8_t);
- Descripción: Con esta función, a partir del argumento "valor", se puede inicializar y configurar el ADC de los siguientes modos o valores:
  - ADC_SIMPLE_FAST: para que lea a máxima velocidad (clk/64)
  - ADC_SIMPLE_SLOW: a velocidad media (clk/128)
  - ADC_OFF: Apaga el ADC. Útil para reducir el consumo
- Observaciones: Existen una gran variedad de configuraciones para este periférico a partir de sus registros; sin embargo se ha diseñado esta función, pues resume los principales usos dentro de este proyecto.

Lectura de valor del ADC

- Función**:** ADC_read(valor_mux, valor_ref)
- Prototipo uint8_t ADC_read(uint8_t, uint8_t);
- Descripción: devuelve un valor de 8bits proporcional al valor analógico leído en función de los siguientes argumentos:
  - "valor_mux": define qué pin del multiplexor (puerto F) ha de ser convertido por el ADC. Toma valores entre 0 y 7
  - "valor_ref" define el valor máximo y por tanto la referencia que debe tener el valor analógico a leer. Puede ser:
    - ADC_5V_REF: referencia interna de 5V
    - ADC_2V_REF: referencia interna de 2,3V
    - ADC_EXT_REF: referencia externa (en pin AREF)
- Observaciones: Véase que a partir del valor de referencia se calculará el resultado final de la conversión. Para este proyecto, como se verá más adelante, sólo se utilizará la referencia interna de 5V y la externa.

3.2.4. Potenciómetro

a. Introducción

El potenciómetro es un elemento electro-mecánico utilizado en electrónica generalmente para el calibrado manual de otros elementos de un circuito, dadas las tolerancias, envejecimiento y ajustes en según qué circunstancias. Para los propósitos de este proyecto es un elemento ideal para la prueba del ADC.

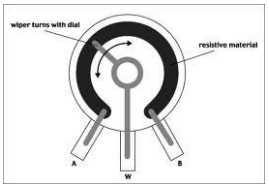

Figura 60. Ejemplo de potenciómetro

b. Funcionamiento eléctrico

Como se puede ver en la imagen de la figura 60, es una resistencia circular que presenta una tercera "pata" que comunica a un terminal móvil. A partir de este terminal móvil se divide la resistencia en dos partes.

Para el caso que aplica, si entre los terminales externos se aplican 5V y 0V, se podrá tener una tensión proporcional a la ubicación del terminal, en el rango de 5-0V, independientemente del número de vueltas del potenciómetro y de la resistencia total que tenga

c. Uso en un microcontrolador

Al igual que los pulsadores normalmente abiertos, comentados en un anterior apartado, y como cualquier dispositivo que sea una entrada para el microcontrolador, debe ir protegido, ya que si no se configura debidamente el microcontrolador puede hacer un cortocircuito y dañar el microcontrolador.

Si se quiere proteger, por ejemplo, con una resistencia de 1kOhm, se debe elegir un potenciómetro sustancialmente mayor, por ejemplo, de 100kOhms. Recuérdese que para tener una variación de tensión, el potenciómetro elegido es indistinto y la elección, de uno u otro, sólo está en función de consumo energético.

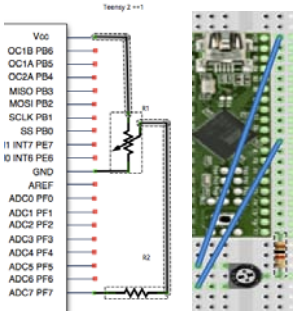

Figura 61. Ejemplos de conexionado de potenciómetro en un microcontrolador

Para tener una referencia visual de los valores tomados del potenciómetro, se podría utilizar una de las funciones de la librería de leds presentada en el anterior apartado, el "medidor LED"

Si, como en la figura 61, se tuviese el acondicionamiento de la resistencia apuntando al pin7 del puerto F (mux 7 del ADC), una función de prueba sería la que se muestra en la figura 62:

```
#include<avr/io.h>  //contiene las definiciones de los registros del microcontrolador

#include"leds.c"

#include"adc.c"

voidmain (void)

{

        ADC_init(ADC_SIMPLE_FAST);                    //configuración sencilla para el ADC

        while (1)                                     //bucle infinito

        {

          leds_meter(ADC_read(7,ADC_5V_REF));                 //iluminará tantos leds como el valor del
ADC con el multiplexor apuntando al 7 y con referencia interna de 5V

        }

}
```

Figura 62. Ejemplo de prueba de potenciómetro con ADC

d. Potenciómetros del proyecto

Para este proyecto se han utilizado dos potenciómetros, que inicialmente sirvieron para probar el uso del conversor analógico digital como los dos ejes del Joystick para el stack USB, pero ya no tienen uso en el proyecto.

Eventualmente podrán ser utilizados como calibradores (digitales) del acelerómetro.

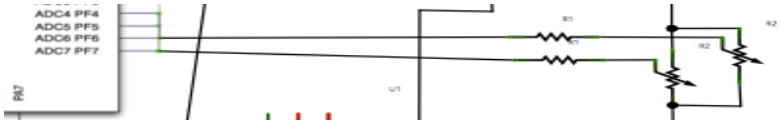

Figura 63. Conexión de los dos potenciómetros al microcontrolador

Para utilizar los potenciómetros en el proyecto se ha utilizado una constante nemotécnica para evitar recordar en que pines estaban conectados. Véase que tendrán el mismo uso que la función ADC_read, ya explicada.

```
#define POT_1                    ADC_read(6,ADC_5V_REF)

#define POT_2                    ADC_read(7,ADC_5V_REF)
```

Figura 64. Fragmento de buttons.h referente a los potenciómetros

3.2.5. Acelerómetro

a. Introducción

Un acelerómetro es un circuito integrado microelectromecánico (MEMS, Microelectromechanical system) capaz de generar una tensión proporcional a la aceleración ejercida en cada eje. La detección de la inclinación se basa en la componente de la gravedad proyectada sobre cada eje al inclinarse.

b. Funcionamiento eléctrico

Existe una gran variedad de modelos, con distintos números de ejes (1-3), distintas tensiones de alimentación continua y distintos interfaces de salida. Algunos ejemplos de saludas pueden ser: salidas analógicas para cada eje, una salida analógica multiplexada para todos los ejes o una salida digital implementada a partir de un interfaz SPI.

Sus principales características son:

- **Encapsulado:** Definirá el tamaño y la conexión a la placa de circuito impreso ("PCB"). Aunque para ser sometido a estudio el empaquetado ideal sería el de "Dual In-line Package" ("DIP"), pues es el empaquetado tradicional con "patillas". Empaquetado preparado para insertar en una Protoboard (placa de inserción) o soldar en una placa perforada. Sin embargo la forma más habitual de este tipo de circuito integrado es la de Land Grid Array ("LGA"), específico para soldadura por ola, que obliga a adquirir un adaptador DIP del fabricante o soldarlo con relativa pericia para su estudio.

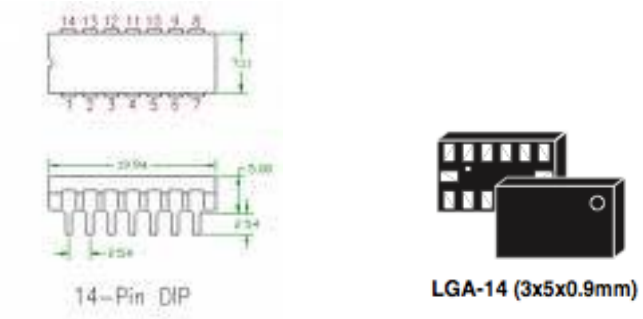

Figura 65. Empaquetados de circuitos integrados

- **Interfaz de salida:** Como se introdujo en el estado de la técnica referente a los acelerómetros (apartado 2.1.3.), el traspaso térmico dentro del integrado está diseñado para que genere una carga eléctrica en función de la aceleración. Sin embargo la salida final por los pines del circuito integrado puede ser una tensión analógica o digital. La salida analógica implica una amplificación y un acondicionamiento para conseguir linealidad entre la aceleración y la tensión. La salida digital, además de lo comentado para la analógica, implica un acondicionamiento de frecuencia, una traducción a valores digitales y un buffer con registro de desplazamiento para ser transmitidos en serie mediante, generalmente, buses SPI e I2C.

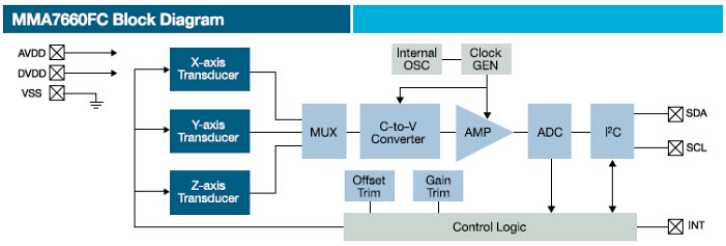

Figura 66. Esquema de funcionamiento de acelerómetro

- **Tensión de alimentación, consumo y función reposo.** Relacionado con el encapsulado del integrado y con la tecnología usada por el fabricante. Es un factor muy importante para el diseño del circuito completo. No obstante, la mayoría de estos circuitos integrados trabajan con rangos tensiones comprendidas entre los 2.6 y 3.3 V, aunque su consumo y sus funciones de reposo pueden variar sensiblemente.

- **Aceleración medible.** La aceleración medible determina los valores máximos que es capaz de registrar el integrado antes de llegar a la saturación. Estos valores están expresados en "g"s y representan valores de aceleración respecto al de la gravedad. Por tanto si un sensor tiene un rango de ±2g implicará que es capaz de medir aceleraciones de hasta 19,6m/s^2 en los dos sentidos.

- **Número de ejes.** Es el número de ejes cartesianos que es capaz de medir el circuito integrado. Un sensor de un solo eje sólo podrá registrar variaciones de aceleraciones en una dirección, uno de dos ejes las podrá registrar en un plano y uno de tres en un espacio tridimensional.

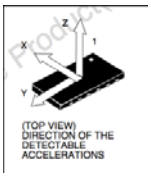

Figura 67. Representación de ejes de un acelerómetro

- **Sensibilidad.** La sensibilidad representa la ganancia del sensor por cada 1g de aceleración que se le aplica en su dirección.

- **Nivel cero-g.** Teniendo en cuenta que las aceleraciones son medidas con respecto al vector de la gravedad, en función de los ejes que tenga, la gravedad siempre será captada por el circuito integrado. De tal forma que, por ejemplo en el caso de estudio de este proyecto, con un acelerómetro de 3 ejes en posición horizontal, el eje Z mostrará un nivel de tensión proporcional a 1g, y este es el nivel cero g ("Zero-g Level").

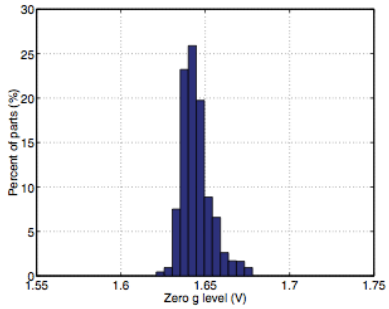

Figura 68. Representación estadística del nivel cero-g

El modelo elegido es el LIS302ALB de ST. Es un acelerómetro de 3 ejes capaz de medir hasta 2g (2 veces la aceleración de la gravedad), con salida analógica y una tensión de alimentación de 3.3V.

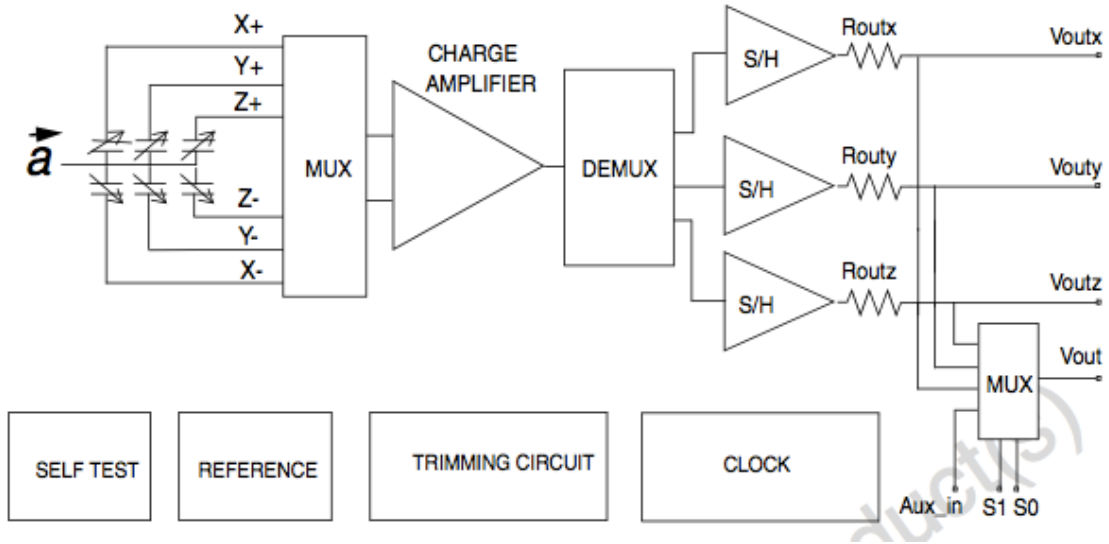

Figura 69. Esquema de funcionamiento eléctrico del acelerómetro del proyecto

La salida analógica puede ser seleccionada bien multiplexada, bien por separado, aunque para este proyecto se utilizarán las tres salidas independientes. Véase que a diferencia de otros dispositivos a leer por el microcontrolador, las salidas de este circuito integrado van protegidas por resistencias (R_{out}) por lo que no serán necesarias otras resistencias de protección

Nótese que la alimentación de este circuito integrado es de 3.3V, no 5V, como el resto del circuito, por lo que habrá que utilizar un regulador de tensión (un circuito integrado LM317) para alimentarlo. El esquema propuesto es el más sencillo propuesto por el fabricante, pues se asume que la alimentación de 5V será estable y filtrada:

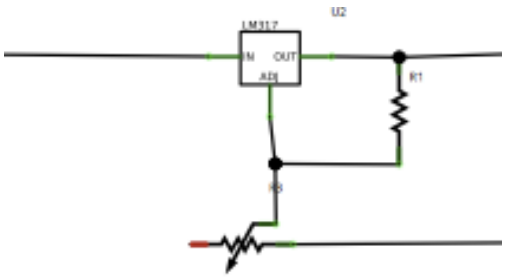

Figura 70. Regulador de tensión para adaptar al acelerómetro

Siguiendo la fórmula proporcionada por el fabricante:

$$Vout=1.25V(1+Rpot/R)$$

Si se toma una resistencia de 1Kohms y un potenciómetro multivuelta de 2Kohms se tiene que se pueden obtener tensiones de salida de 1.25V hasta 3.75V. En la figura 40, al inicio de este apartado, se puede ver cómo y dónde se instalaron estos componentes en la placa de pruebas del proyecto.

Como ya se adelantó al inicio del apartado 3(figura 39), el encapsulado del acelerómetro elegido está pensado para ser soldado por ola sobre una PCB y no para ser utilizado en una perfboard, como es el caso de este proyecto. Por ello, se ha soldado a un zócalo de manera artesanal. No se recomienda esta opción dada su elevada dificultad y peligro para el acelerómetro.

c. Uso en un microcontrolador

Tras adaptar la alimentación, las tres tensiones de salida del acelerómetro se conectarán a tres puertos del multiplexor ADC, es decir el puerto F, pines 3,4 y 5.

También se deberá medir con la tensión de alimentación de 3.3V, como referencia externa, para el acelerómetro. Para el ejemplo de un solo eje, conectado al pin 3 del ADC, el código sería similar al usado por el potenciómetro:

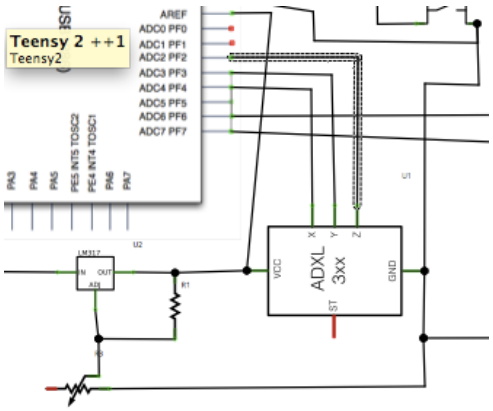

Figura 71. Conexión del acelerómetro a un microcontrolador

```
#include<avr/io.h>  //contiene las definiciones de los registros del microcontrolador

#include"leds.c"

#include"adc.c"

voidmain (void)

{

        ADC_init(ADC_SIMPLE_FAST);                    //configuración sencilla para el ADC

        while (1)                                     //bucle infinito

        {

          leds_meter(ADC_read(3,ADC_EX_REF));                 //iluminará tantos leds como el valor del
ADC con el multiplexor apuntando al 3 y con referencia externa

        }

}
```

Figura 72. Ejemplo de acelerómetro que enciende tantos leds como el valor proporcionado por un eje del acelerómetro.

d. Acelerómetros del proyecto

Con vistas a ser la palanca de un Joystick, el valor final debe de estar comprendido entre -100 y 100 para cada eje, lo que implica inexorablemente que hay que cambiar no sólo el valor de salida del ADC sino también el tipo de dato, que se deberá pasar de entero sin signo a entero con signo.

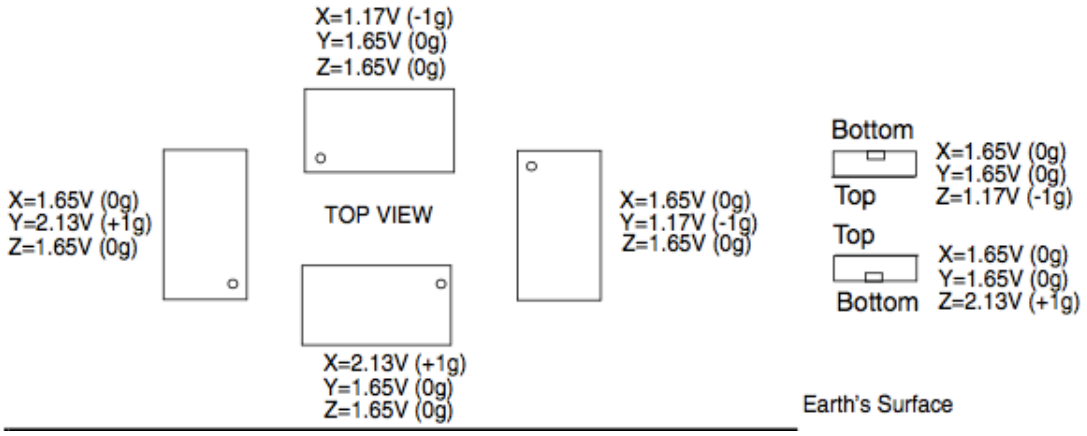

Figura 73. Medidas analógicas en función de la posición del acelerómetro

Siguiendo las especificaciones del fabricante, teniendo en cuenta que la referencia del ADC serán los 3.3V de alimentación del circuito integrado y que se tiene una salida comprendida entre 0 y 255 (8 bits), se puede establecer la siguiente escala.

0 --------- -100

255 --------- 100

Al valor obtenido del ADC se le ha de restar la mitad del valor máximo (128) y dividirlo entre 1.27, así como encapsularlo en una variable que permita números negativos. De esta forma los valores obtenidos estarán en el rango de -127 hasta 127.

No obstante estos cálculos no son lo más ideal en términos de confort, pues para alcanzar el máximo valor (100) no sólo basta con inclinar el acelerómetro, pues sólo se obtendría un máximo de 1g (50 unidades), y haría falta "agitar" el mando para llegar a esta aceleración.

Si se pretende alcanzar el máximo valor con 1g y que para el resto de valores superiores, sature en 100, entonces el valor a dividir será 0,64 y se deberá imponer una condición tal que para valores mayores de 100 o menores de -100 tome ese valor.

Las funciones de control de los acelerómetros están incluidas en la librería buttons

```
// Fragmento de buttons.h

#define JOY_CONFIG                ADC_init(ADC_SIMPLE_FAST)

#define JOY_X                     ADC_read(3,ADC_EX_REF)

#define JOY_Y                     ADC_read(4,ADC_EX_REF)

//#define JOY_Z                   ADC_read() No se usa

//=========================================================================

// Funicones

//=========================================================================

int8_t joy_adquire_X();        // Devuelve un valor comprendido entre -100 y 100

int8_t joy_adquire_Y();
```

Figura 74. Fragmento de la librería buttons relacionado con el acelerómetro

Tras configurar los pines del ADC interno del microcontrolador a los que está conectado cada eje del acelerómetro, mediante los "defines" JOY_X y JOY_Y, (ver apartado ADC) se pueden usar las funciones:

Adquisición de movimientos en el eje X

- Función**:** joy_adquire_X();
- Prototipo int8_t joy_adquire_X();
- Descripción: Configura el ADC al que está conectado el ADC y devuelve un valor comprendido entre -100 y 100 en función de la posición del acelerómetro
- Observaciones: Se ha calibrado esta función para que a partir de 1G de gravedad en un eje (hasta 2G) sature la función y ofrezca un valor de 100, o -100 en función de la dirección.

Adquisición de movimientos en el eje Y

- Función**:** joy_adquire_Y();
- Prototipo int8_t joy_adquire_Y();
- Descripción: Análoga a la anterior función pero para el eje Y

3.2.6. Generación de modulación de ancho de pulso.

a. Introducción

Al igual que el ADC es útil para captar elementos que miden el entorno a partir de señales analógicas, la generación de pulso de ancho modulado (Pulse Width Modulation, PWM) es útil para accionar elementos eléctricos analógicos, como motores, sonidos y otras señales analógicas.

b. Funcionamiento eléctrico

A partir de la modulación de ancho de pulso se genera una señal cuadrada en la que se controla la frecuencia y el ciclo de trabajo. Como se puede ver en la figura 75, con una misma frecuencia, la tensión efectiva (V average) disminuye al disminuir el ciclo de trabajo. Esto es útil, por ejemplo, para controlar la potencia de motores eléctricos.

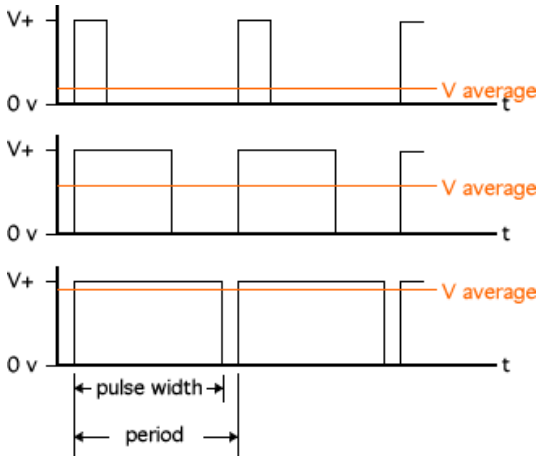

Figura 75. Imagen que ilustra el cambio del ciclo de trabajo para una misma frecuencia

La modulación del ancho de pulso se utiliza también, a partir de un acondicionamiento adecuado, para la generación de señales analógicas, como se puede ver en la figura 76.

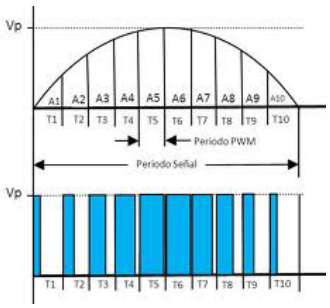

Figura 76. Teoría de generación de señal senoidal a partir de onda cuadrada modulada

c. Uso en un microcontrolador

De manera análoga a un ADC, un PWM tienen salidas multiplexadas para multiplicar el número de salidas, pero a diferencia del ADC, no se consiguen salidas independientes, pues todos los puertos multiplexados comparten cierta configuración y, mientras continúa la ejecución de la pila del programa, se sigue emitiendo por esas salidas.

Para entender mejor el uso de un PWM por parte de un microcontrolador, se puede idealizar como un elemento que una vez se inicia, se ejecuta de manera paralela e independiente al programa. Es un periférico que lleva una cuenta de ciclos de reloj y que, según su configuración, activa o desactiva la salida a distintos ritmos.

Por otro lado, se debe entender también que la generación de un pulso de ancho modulado, el microcontrolador tiene dos formas de trabajo principales: la "estándar" y la "corregida". La diferencia se basa en cuándo empieza el ciclo de trabajo: en el estándar empieza cuando empieza el ciclo, y en el corregido procura tomar los valores centrales del ciclo. Esta diferenciación es importante cuando se planea someter la salida a cambios en el ciclo o la frecuencia, pues la forma corregida consigue el cambio más "suave" a costa de dividir la frecuencia máxima entre dos. Esta diferencia se puede apreciar de manera gráfica en la figura 77.

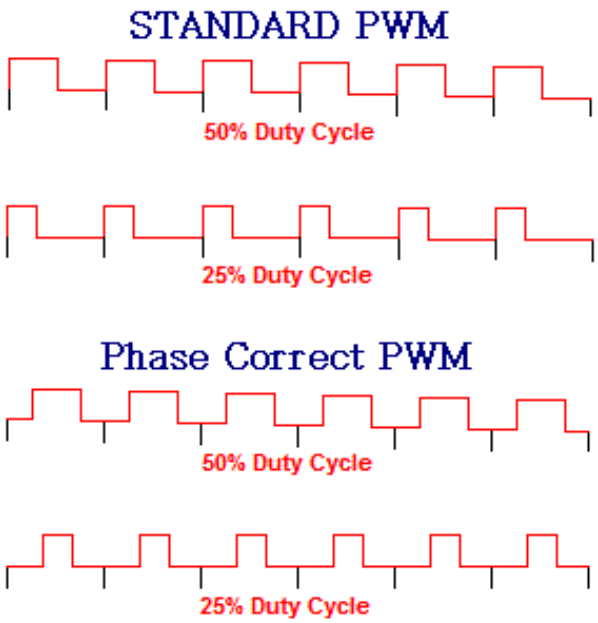

Figura 77. Diferencias entre PWM estándar y corregido

Nota: En este proyecto sólo se trabajará con "phase correct PWM".

Para controlar su uso se utilizan los siguientes registros (véase que cuando un dato es sustituido por una "X" se refiere al número del PWM, y si es por "XX" se referirá al número del PWM y a la letra del puerto, respectivamente).

- **TCCR"X"C:** sólo disponible en los PWM de 16bits, para usar el puerto de PWM con otros fines
- **TCCR"X"B:** Selecciona la escala del reloj. Con este registro se selecciona el número de ciclos de reloj que debe hacer el PWM para aumentar su cuenta. Divide por tanto la frecuencia de reloj a efectos de cálculo para el PWM. Recuérdese que según esta memoria, se trabaja con PWM corregidos, que dividen de por sí la frecuencia de reloj entre dos.
- **ICR"X"L (H):** Preselecciona un número desde el que empezar a contar. Se utiliza para variar la frecuencia del pulso. Para 16 bits, los 8 más significativos irán en ICRxH y para los 8 menos significativos irán en ICRxL
- **OCR"XX":** Con este registro se varía el ciclo de trabajo del pulso
- **TCCR"X"A**: Selecciona el modo de operación del PWM (corregido, rápido, estándar)

Para probar los conceptos se puede utilizar un led conectado al puerto B7 que coincide con el PWM 0A. Para que a través del PWM se varíe cada 4ms la intensidad del led de forma cíclica.

```
#include<avr/io.h>
#include<util/delay.h>
Voidmain (void)
{
        unsignedchar i = 0;
        DDRB = (1<<7);                  // Puerto del led/pwm para salida
        TCCR0A = (1<<COM0A1)|(1<<WGM00); // modo CLK/8
        TCCR0B = (1<<CS00);             // modo rápido (no corregido)
        while (1)
        {
          for(i=0;i<255;i++)// cuenta ascendente
          {
                OCR0A = i;              // se varía el ciclo de trabajo en función de la cuenta
                _delay_ms(4);
          }
          for(i=255;i>0;i--)      // cuenta descendente
          {
                OCR0A = i;              // varía el ciclo/intensidad led en función de la cuenta
                _delay_ms(4);           // retardo 4ms
          }
        }
}
```

Figura 78. Ejemplo de código para ejecutar un PWM

d. PWM del proyecto

En el caso del microcontrolador de este proyecto se dispone de 4 PWM, dos de 8 bits con dos salidas y otros dos de 16 bits con tres salidas.

```
//=====================================================================
// Configuraciones
//=====================================================================
// Recordatorio |= para poner PIN como salida
#define PWM_PIN_0A      DDRB |= (1<<7)          //8bit only
#define PWM_PIN_0B      DDRD |= (1<<0)          //8bit only
#define PWM_PIN_1A      DDRB |= (1<<5)
#define PWM_PIN_1B      DDRB |= (1<<6)
#define PWM_PIN_1C      DDRB |= (1<<7)
#define PWM_PIN_2A      DDRB |= (1<<4)          //8bit only
#define PWM_PIN_2B      DDRD |= (1<<1)          //8bit only
#define PWM_PIN_3A      DDRC |= (1<<6)
#define PWM_PIN_3B      DDRC |= (1<<5)
#define PWM_PIN_3C      DDRC |= (1<<4)
//=====================================================================
// Argumentos nemotécnicos para las funciones
//=====================================================================
// PWM_correct_init() "port"  cases
#define PWM_0A  0x0A            //8bit only
#define PWM_0B  0x0B            //8bit only
#define PWM_1A  0x1A
#define PWM_1B  0x1B
#define PWM_1C  0x1C
#define PWM_2A  0x2A            //8bit only
#define PWM_2B  0x2B            //8bit only
#define PWM_3A  0x3A
#define PWM_3B  0x3B
#define PWM_3C  0x3C

// PWM_correct_init() timerdivision (remember "correct PWM" always divides clk/2)
#define PWM_CLK_N               0x01
#define PWM_CLK_8               0x02
#define PWM_CLK_64              0x03
#define PWM_CLK_256     0x04
#define PWM_CLK_1024    0x05
//=====================================================================
// definiciones internas para no dar mal uso accidental a los registros
//=====================================================================
#define TIMER_MASK              (0x07&timer)
//Para pasar de 16bits a 8bits y que se quede la parte más significativa
#define PHASE_8B        ((uint8_t)(phase>>8))
#define DUTY_8B                 ((uint8_t)(duty>>8))
//=====================================================================
// Funciones
//=====================================================================
voidPWM_correct(uint8_t, uint8_t, uint16_t, uint16_t);      //genera un PWM corregido a partir de (puerto,
prescaler de clk, inicio de cuenta, ciclo de trabajo)
voidPWM_correct_duty(uint8_t, uint16_t);        //modifica el PWM corregido a partir de (puerto, ciclo)
voidPWM_correct_freq(uint8_t, uint8_t, uint16_t); //modifica el PWM corregido a partir de (puerto, división del
timer,inicio de cuenta)
```

Figura 79. Fragmento de la librería de uso del PWM

Tras configurar los puertos en los que están ubicados los PWM, se pueden usar las funciones que simplifican el uso de PWM corregidos para este proyecto:

Generación de señal de ancho de pulso modulado

- Función**:** PWM_correct(puerto, divisor_de_reloj, inicio_de_cuenta, ciclo_de_trabajo)
- Prototipo void PWM_correct(uint8_t, uint8_t, uint16_t, uint16_t)
- Descripción: Inicia una señal cuadrada PWM corregida en función de los argumentos utilizados
- Observaciones: De todas las funciones creadas, es la única que inicia correcta y completamente esta señal.

Variación de ciclo de trabajo de una señal PWM

- Función**:** PWM_correct_duty (puerto, ciclo_de_trabajo)
- Prototipo void PWM_correct_duty (uint8_t, uint16_t);
- Descripción: Varía el ciclo de trabajo de una señal PWM
- Observaciones: La señal ha de ser previamente creada por la función PWM_correct(,,,).

Variación de la frecuencia de una señal PWM

- Función**:** PWM_correct_freq (puerto, divisor_de_reloj, inicio_de_cuenta)
- Prototipo void PWM_correct_freq(uint8_t, uint16_t, uint16_t)
- Descripción: Varia la frecuencia de una señal previamente generada por PWM_correct()
- Observaciones: Para entender el funcionamiento de esta función es necesario entender la función de sus argumentos.

Los argumentos comunes son:

- **puerto**: puede tomar los siguientes valores en función del pin que se quiera utilizar. Véase que son PWM tanto de 16 bits como de 8 bits.
  - **PWM_Xx**: donde “X” indica el PWM y “x” el pin de salida para cada PWM
- **divisor_de_reloj**: Divide la frecuencia de reloj del microcontrolador (clk) entre valores fijos. Véase que al tratarse de PWM corregidos, el número que divide según la definición, siempre es multiplicado por 2.
  Resulta interesante para dividir la frecuencia entre números enteros:
  - **PWM_CLK_N**: Divide la frecuencia de reloj entre 2
  - **PWM_CLK_8**: Divide la frecuencia de reloj entre 16
  - **PWM_CLK_64:** Divide la frecuencia de reloj entre 128
  - **PWM_CLK_256**: Divide la frecuencia de reloj entre 512
  - **PWM_CLK_1024**:Divide la frecuencia de reloj entre 2048

- **inicio_de_cuenta**: Teniendo en cuenta que la frecuencia que se desee generar estará comprendida entre dos valores proporcionales a la frecuencia de reloj, se puede demorar el inicio de la cuenta para generar la onda y así alcanzar valores que no son fracciones enteras del reloj. En función del PWM utilizado, este valor puede ser de 8 bits o de 16. Sin embargo la función está preparada para aceptar siempre un valor de 16 bits y convertirlo si es necesario
- **ciclo_de_trabajo**: como se ha explicado en la teoría, varía el ciclo de trabajo. Al igual que el argumento inicio_de_cuenta, puede tomar valores de 8 y 16 bits, aunque acepta valores de ambos y lo convierte al que proceda en función de si se está utilizando un PWM de 8 o 16 bits.

3.2.7. Sonidos.

a. Introducción

Una forma de comprobar con precisión el uso de PWM es mediante la generación de sonido, pues mediante la frecuencia se generan las distintas notas y variando el ciclo de trabajo se puede variar el volumen.

b. Funcionamiento eléctrico

El sistema occidental de notas se derivan del poema “Ut queant laxis” del monje benedictino friulano Pablo el Diácono, que divide las freciencias en 7 notas, con definiciones intermedias como bemoles y sostenidos, y a su vez en octavas, que son sus múltiplos enteros.

Nota\Octava	1	2	3	4	5	6	7
Do	32,7	65,41	130,81	261,63	523,25	1046,5	2093
Re	36,71	73,42	146,83	293,66	587,33	1174,66	2349,32
Mi	41,2	82,41	164,81	329,63	659,26	1328,51	2637,02
Fa	43,65	87,31	174,61	349,23	698,46	1396,91	2793,83
Sol	49	98	196	392	783,99	1567,98	3135,96
La	55	110	220	440	880	1760	3520
Si	61,74	123,47	246,94	493,88	987,77	1975,53	3951,07

Como se puede ver, para cada nota de la primera octava, la segunda octava resulta de multiplicar por dos la frecuencia inicial. Y así, sucesivamente, con el resto de octavas.

El sonido es producido por un altavoz (bocina o buzzer), que es un transductor electromecánico de doble sentido. Es decir, es un elemento capaz de transformar una excitación eléctrica en una mecánica y viceversa, que podría actuar tanto como micrófono como altavoz.

La forma en la que actúa la parte mecánica del altavoz se puede entender como una pequeña bomba muy rápida que varía la presión del aire para así producir el sonido. Dada esta analogía, si se varía la frecuencia con la que vibra se obtienen las diferentes notas, y si se varía el ciclo de trabajo, se varía la potencia de salida del sonido.

En función de si se trata de un altavoz o de un piezoeléctrico, se deberá acondicionar o no. En este caso de estudio, se utilizará el altavoz acondicionado y activo por nivel bajo.

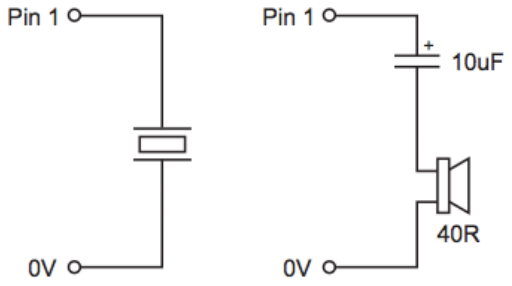

Figura 80. Ejemplos de conexión de un buzzer a un microcontrolador

c. Uso en un microcontrolador

Para generar notas con un microcontrolador, basta con generar un PWM y variar su registro de cuenta de tiempo ICRxH/L para la frecuencia, y OCRxx para variar su volumen.

Como ejemplo se pueden usar los dos potenciómetros conectados a las entradas ADC del microcontrolador, para variar tanto la frecuencia como el ciclo de trabajo, y las funciones explicadas en el apartado PWM, para crear un ciclo corregido.

Véase que se trata de un PWM de 16 bits y el ADC está programado para dar salida de 8 bits, por lo que habrá que realizar un desplazamiento de 8 bits hacia la izquierda (con el operador "<<8" en la figura 82), con lo que se perderá resolución, pero servirá a los propósitos de este ejemplo.

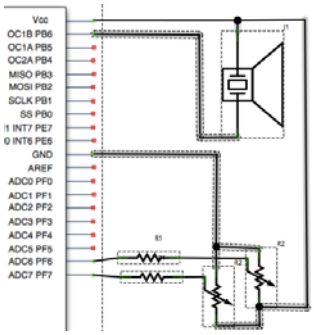

Figura 81. Conexión de un buzzer y dos potenciómetros a un microcontrolador

```
#include<avr/io.h>
#include"pwm.c"
#include"adc.c"
voidmain (void)
{
        ADC_init(ADC_SIMPLE_FAST);                  //configuración sencilla para el ADC
        PWM_correct(PWM_1B, PWM_CLK_N, 0,0)         //inicia un PWM corregido

        while (1)
        {
        PWM_correct_duty(PWM_1B, ADC_read(6,ADC_EX_REF)<<8);          // varia volumen
        PWM_correct_freq(PWM_1B, PWM_CLK_N, ADC_read(7,ADC_EX_REF)<<8);// varia
frecuencia
        }
}
```

Figura 82. Ejemplo de código para probar un buzzer a partir de PWM

Si se quisiese diseñar una función que trasladase una frecuencia en hercios a un valor para el registro de cuenta del PWM, se deberá tener el valor en coma flotante (para mayor precisión), tener en cuenta que los ciclos PWM utilizados son corregidos y calcular siguiendo la siguiente fórmula:

"frecuencia deseada" ="mitad frecuencia reloj" / "valor contador"

d. Buzzer del proyecto

Sigue un sencillo esquema electrónico similar al usado en el ejemplo.

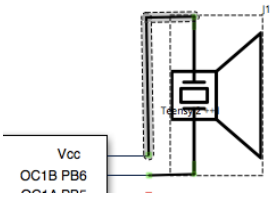

Figura 83. Conexión del Buzzer en el proyecto

Las funciones diseñadas para este periférico consideran entradas de 8 bits aunque el dispositivo sea de 16 bits

```
//=====================================================================
// Configuraciones
//=====================================================================
#define BUZZ_PWM_PORT   PWM_2B
#define HALF_SYS_CLK    8000000         //la mitad de la freq del reloj

//=====================================================================
// Function defines
//=====================================================================
//valores de las notas en séptima octava (_S para sostenido)
#define DO              2093
#define DO_S            2217.46
#define RE              2349.32
#define RE_S            2489.02
#define MI              2637.02
#define FA              2793.83
#define FA_S            2959.96
#define SOL             3135.96
#define SOL_S           3322.44
#define LA              3520
#define LA_S            3720.31
#define SI              3951.07

//=====================================================================
// Functions
//=====================================================================
uint16_t buzz_phase(float, uint8_t);  //genera un contador a partir de nota,octava (interno)
uint16_t byte2volume (uint8_t);       // pasa de 8b a 16bits (interno)
voidbuzz_init(uint8_t, float, uint8_t);   //inicia con una nota
voidbuzz_volume (uint8_t);                //modifica volumen
voidbuzz_note(float, uint8_t);            //modifica sonido (nota, octava)
```

Figura 84. Fragmento de librería de control del buzzer

Tras haber configurado el puerto PWM donde está conectado el nivel bajo del buzzer, se puede usar las funciones que se apoyan en la librería PWM:

Inicialización del altavoz:

- Función**:** buzz_init(volumen, nota, octava)
- Prototipo void buzz_init(uint8_t, float, uint8_t)
- Descripción: Inicia el buzzer generando una nota en función de la octava y del volumen seleccionado en los argumentos.
- Observaciones: El valor del volumen debe estar comprendido entre los valores [0-0xFF], la nota debe tomar un valor equivalente en frecuencia a la nota de la séptima octava, y la octava un valor comprendido entre [1-7]

Control de volumen del altavoz:

- Función**:** buzz_volume(volumen);
- Prototipo: void buzz_volume(uint8_t)
- Descripción: Varía el volumen en función del valor de 8bits. 0 para el mínimo, 255 para el máximo volumen.
- Observaciones: Esta función está diseñada para ser utilizada tras haberse inicializado el altavoz.

Cambio de nota del altavoz:

- Función**:** buzz_note(nota, octava)
- Prototipo: void buzz_note (float, uint8_t)
- Descripción: varía la nota y la octava a reproducir, sin variar el volumen.
- Observaciones: la nota debe tomar un valor equivalente en frecuencia a la nota de la séptima octava, y la octava un valor comprendido entre [1-7].

Conversión de un byte a volumen:

- Función: byte2volume (byte)
- Prototipo: uint16_t byte2volume(uint8_t);
- Descripción: Una función de uso interno que permite pasar de un valor de 8bits dado como volumen a un valor válido para un ciclo de trabajo
- Observaciones: Se ha diseñado esta función para simplificar los argumentos de entrada de las funciones públicas del altavoz, puesto que el máximo valor de volumen se alcanza con un 50% del ciclo de trabajo.

Conversión de frecuencia a temporizadores de PWM:

- Función: buzz_phase (nota, octava);
- Prototipo: uint16_t buzz_phase(float, uint8_t);
- Descripción: Función de uso interno que a partir de el valor de una nota y de su octava calcula el valor de los registros para utilizar el PWM

Véase que las notas, en la definición (o “define”) están expresadas en la séptima octava, pues resulta más sencillo calcular con precisión octavas menores a partir de este dato.

3.3. Interfaz

3.3.1. Introducción

El stack USB LUFA es muy extenso y complejo, y está diseñado para que se le adapten las librerías y no al revés. Dado el planteamiento de este proyecto, que parte de los periféricos para acabar en un USB, se requiere reingeniería para poder adaptar las librerías.

Tras el estudio de la arquitectura de LUFA, se requiere la modificación de determinados archivos, que se encuentran en 2 rutas:

- LUFA/Drivers/Board
- Demos/Device/ClassDriver/Joystick

En la primera ruta se deberán incluir las librerías creadas. Igualmente, hay que adaptar las que allí existen para que actúen como interfaz, ya que en ellas se encuentran los controladores de los LEDs, Botones y Joystick que utiliza LUFA

En la ruta “Demos/Device/ClassDriver/Joystick”, se encuentran los archivos de los descriptores y las funciones principales del dispositivo, así como el Makefile a configurar

3.3.2. Makefile

El archivo Makefile es una modificación del creado por Eric B.Weddington, pero con una línea a la que hay que prestar especial atención, la línea:

<BOARD = USER>

Al definir esta línea le estamos especificando al stack y al compilador que las librerías que usa los periféricos de la placa se encuentran en la ruta /Board/Board.

3.3.3. Drivers

La forma más sencilla de asegurar que el driver que se va a utilizar contiene todas las funciones que va a reclamar el stack, es duplicando la carpeta /Board/USBKEY, y renombrándola a /Board/Board, pues es dónde hemos especificado que se encuentra nuestras librerías de control de periféricos.

Para ver cómo quedaría esta carpeta para que pueda ser utilizada, véase código adjunto al proyecto.

3.3.4. Configuración del dispositivo y sus descriptores

Para poder adaptar las librerías creadas, y realizar funciones no contempladas por el stack del joystick, se deben editar los archivos Joystick.c y Joystick.h.

Tal y como está programado, en el archivo Joystick.c se pueden incluir las inicializaciones de los periféricos para que se ejecuten cuando se inicia (se conecta físicamente) el dispositivo. En esta función se puede añadir la inicialización del sonido "buzz_init(0,DO,1)" que iniciará el sonido en volumen 0, y en DO en primera octava

```
/** Configures the board hardware and chip peripherals for the demo's functionality. */
void SetupHardware(void)
{
	/* Disable watchdog if enabled by bootloader/fuses */
	MCUSR &= ~(1<< WDRF);
	wdt_disable();

	/* Disable clock division */
	clock_prescale_set(clock_div_1);

	/* Hardware Initialization */
	Joystick_Init();
	LEDs_Init();
	Buttons_Init();
	USB_Init();
	buzz_init(0, DO, 1);   //← Línea añadida
}
```

Figura 85. Detalle de función de inicialización editada

Como se ha comentado en el apartado 2.14 (HID-USB), se establece una comunicación por interrupciones, mediante la cual el sistema informático reclama al dispositivo USB un informe con los datos de estado del Joystick. Es en este momento en el que se puede aprovechar la función activa para utilizar las funciones de las librerías diseñadas, como por ejemplo, para generar sonidos en función de los botones presionados, el volumen o el nivel de LEDs activados

Puesto que el stack del USB como Joystick espera una serie de cursores digitales para dar valores analógicos y, este proyecto usa un sistema analógico, se debe adaptar a las librerías creadas.

La función modificada que contemple el uso de un sistema analógico como palanca de mando del joystick y para generar sonidos en función de los botones creados, sería:

```
bool CALLBACK_HID_Device_CreateHIDReport(USB_ClassInfo_HID_Device_t* const HIDInterfaceInfo,uint8_t* const
ReportID,const uint8_t ReportType,void* ReportData, uint16_t* const ReportSize)

{
        USB_JoystickReport_Data_t* JoystickReport = (USB_JoystickReport_Data_t*)ReportData;
        uint8_t JoyStatus_LCL    = Joystick_GetStatus();
        uint8_t ButtonStatus_LCL = Buttons_GetStatus();

        /** Inicio de lineas añadidas/modificadas. */
        JoystickReport->Y = joy_adquire_Y();
        JoystickReport->X =  joy_adquire_X();
        JoystickReport->Button = Buttons_GetStatus();
        switch (JoystickReport->Button)
        {
          case1: buzz_note(DO, 3); break;
          case2: buzz_note(RE, 3); break;
          case3: buzz_note(MI, 3); break;
          default: buzz_note(0, 1); break;
        }
        leds_meter(POT_1);
        buzz_volume(POT_2);
        /** Fin de líneas añadidas/modificadas. */

        *ReportSize = sizeof(USB_JoystickReport_Data_t);
        return false;
}
```

Figura 86. Detalle de función Joystick.c editada

También se podría editar el archivo Descriptors.c para añadir un nombre y un autor personalizado al Joystick a la hora de enumerar:

```
/** Manufacturer descriptor string. This is a Unicode string containing the manufacturer's
details in human readable
 *  form, and is read out upon request by the host when the appropriate string ID is
requested, listed in the Device
 *  Descriptor.
 */
USB_Descriptor_String_t PROGMEM ManufacturerString =
{
        .Header                 = {.Size = USB_STRING_LEN(11), .Type = DTYPE_String},

        .UnicodeString          = L"Felix Gomez"
};

/** Product descriptor string. This is a Unicode string containing the product's details in
human readable form,
 *  and is read out upon request by the host when the appropriate string ID is requested,
listed in the Device
 *  Descriptor.
 */
USB_Descriptor_String_t PROGMEM ProductString =
{
        .Header                 = {.Size = USB_STRING_LEN(18), .Type = DTYPE_String},

        .UnicodeString          = L"PFC CarlosIII"
};
```

Figura 87. Detalle de edición de Descriptors.h

3.4. Preparación del proyecto

En este apartado se especificarán las herramientas y la configuración para el desarrollo del proyecto, así como unos últimos conceptos a modo de extensión del estado de la técnica.

3.4.1. Makefile

Aunque no es necesario para pruebas sencillas, resulta muy útil, e incluso obligatorio para programas complejos, conseguir un archivo makefile para el compilador gcc-avr. Es un archivo que especifica una gran variedad de opciones y simplifica el comando para compilar el código.

En este archivo de texto es necesario editar ciertas líneas para adecuarlo al hardware y al archivo resultante que se quiere obtener.

En este ejemplo se busca compilar el archivo “prueba.c” en el binario “prueba.hex”

```
# Target file name (withoutextension).

TARGET = prueba

# List C source files here. (C dependencies are automaticallygenerated.)

SRC =  prueba.c

# MCU name, you MUST set thisto match theboardyou are using

# type "makeclean" afterchangingthis, so all files will be rebuilt

MCU = at90usb1286      # Teensy++ 2.0

# Processorfrequency.

#   Normallythefirstthingyourprogramshould do is set theclockprescaler,

#   so yourprogramwillrun at thecorrectspeed.  Youshouldalso set this

#   variable tosameclockspeed.  The _delay_ms() macro uses this, and many

#   examples use this variable tocalculatetimings.  Do notadd a "UL" here.

F_CPU = 16000000
```

Figura 88. Ejemplo de líneas a editar en archivo makefile

El archivo más utilizado para microcontroladores Atmel es el diseñado para el winAVR por Eric B.Weddington, Jorg Wunsch y otros colaboradores. Sin embargo, para la compilación de todo el proyecto, se ha utilizado el propio del stack USB “LUFA” debidamente configurado, que es en esencia el propuesto pero adaptado para una mejor gestión de las interrupciones propias del usb.

Una vez preparado tanto el código como el makefile, los comandos principales, según el propio makefile, para compilar son:

- **“makeclean”** elimina el rastro de cualquier compilación previa
- **“makeall”** compila todo el programa a partir del archivo especificado.

3.4.2. Creación de fuentes y dependencias

Una vez se tiene el archivo makefile configurado y dispuesto en una carpeta, basta con crear archivos de texto en los que se escriba el código a compilar.

Estos archivos pueden ser creados con cualquier editor de textos, pero es recomendable el uso de editores de código que coloreen las palabras claves para hacer más fácil de seguir el código.

Todo el código desarrollado en este proyecto se ha hecho con el editor de textos que se facilita por defecto en la instalación de Ubuntu: el Gedit.

También se pueden instalar IDEs, o entornos de programación, que además de colorear las palabras claves, las indexan y las muestran como ayuda a la hora de programar. No obstante, para programadores noveles puede resultar demasiado confuso y aparatoso, por lo que no se recomienda su uso.

Para poder compilar el código generado, pues el makefile es solo un script que corre a partir de funciones del sistema operativo, es necesario instalar determinadas librerías en el sistema operativo. Al igual que el resto de paquetes de esta memoria, están disponibles a través del repositorio de Ubuntu.

Estas librerías son:

- **lib-avr:** Contiene librerías utilizadas para programar AVR, como por ejemplo las definiciones de los registros para cada microcontrolador
- **lib-usb dev**: Contiene librerías para la compilación del stack USB

Y el programa compilador para microcontroladores: “avr-gcc”

3.4.3. Programa cargador

Una vez se ha generado el archivo binario compilado, existen dos vías para programarlo en el microcontrolador.

Con la herramienta libre avr-dude, mediante línea de comandos (o configurando un IDE como "eclipse") y con un programador como el USB-ISP-AVR-MK2. Aunque esta forma no será motivo de estudio.

O, como en el caso de la mayoría de los microcontroladores USB, mediante una aplicación dedicada y usando la capacidad Bootloader, si está pre-programado con esta característica.

Para el Teensy que se ha utilizado en este proyecto, en su página web [11] se puede encontrar una aplicación de código libre que, de forma automática, programa el microcontrolador detectado a partir de un archivo binario.

Figura 89. Loader de firmware para Teensy

3.4.4. Bloque Fritzing

Para la presentación de este proyecto se ha escrito el bloque correspondiente al Teensy para el Fritzing, tanto para agilizar su diseño, como para la estética del documento.

Para diseñar un componente utilizable como elemento en Fritzing, se han de crear tres imágenes, una con la vista del elemento para insertar en una protoboard, un esquema lógico con terminales y una representación a escala 1:1 de los conectores a imprimir en la PCB. Una vez dispuestas estas imágenes, se deben especificar los conectores y vincularlos a las tres imágenes. El resultado del bloque añadido se puede ver en la figura 90:

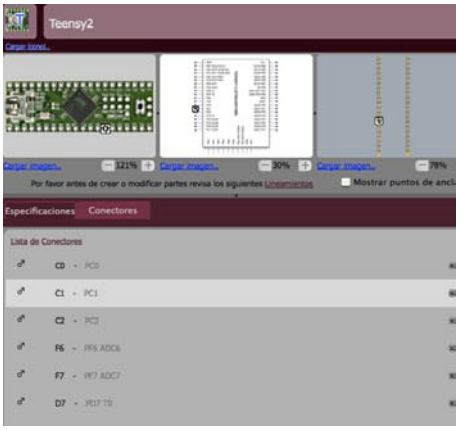

Figura 90. Captura del elemento teensy diseñado para Fritzing

4. Pruebas

Las pruebas para la realización de este proyecto se han planteado de la siguiente forma:

- Prueba programación sobre placa con ATMega 16, encendido y apagado de leds, botones, buzzer y pantalla LCD de 2 líneas y 16 caracteres. También se hicieron pruebas preliminares para el acelerómetro, puertos serie (SPI y TWI) con el IDE Eclipse preparado para AVR y se adaptó para ser programado con el AVR IPS MK2 (figura 91).

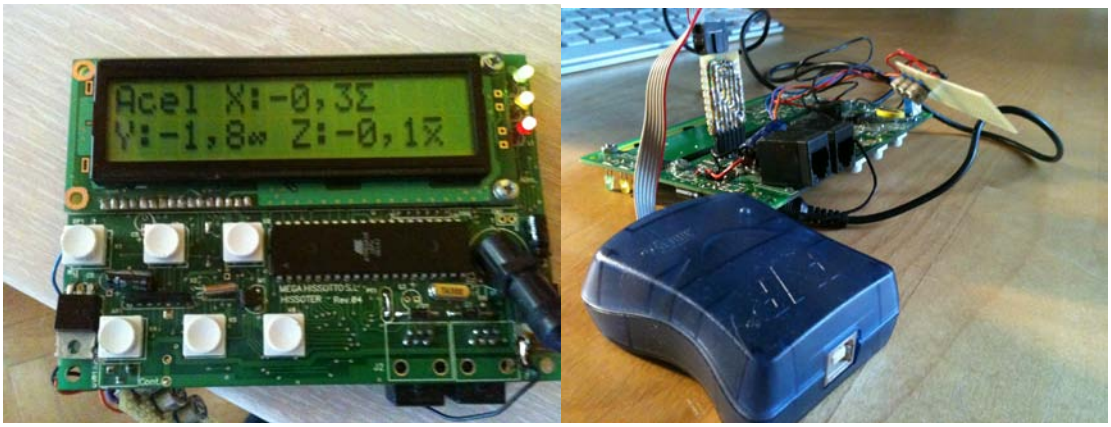

Figura 91. Imágenes de placa de pruebas ATMega16

- Prueba del stack USB "LUFA". Con configuración para micro y reloj utilizado, sin añadidos, sobre software probador de USB en distintas plataformas, MacOSX 10.6, Ubuntu 9.10-11.04 y Windows XP-7. Se ha podido comprobar que los distintos sistemas operativos enumeraban bien el microcontrolador como un HID-USB

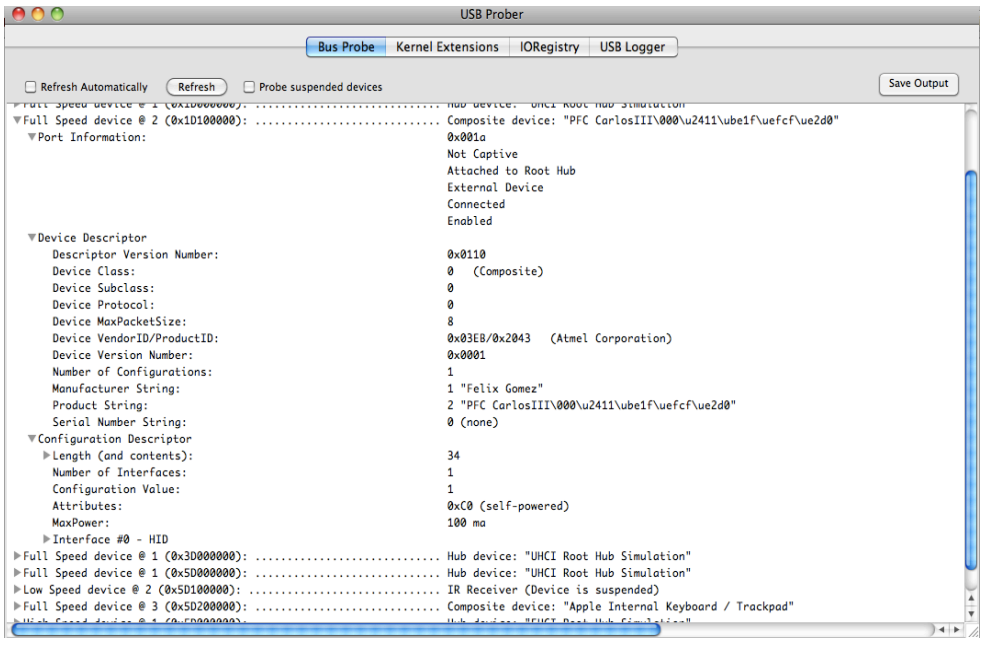

Figura 92. Pruebas de enumeración con "USB Probe" de OSX

- Pruebas de los distintos elementos hardware utilizados:
    - Leds: Encendido y apagado a partir de temporizadores
    - Botones digitales: Tras desarrollar una librería para el control de leds, se desarrolló otra capaz de accionarla a partir de botones en puertos digitales
    - PWM y Speaker. Para pruebas preliminares de la función PWM, se varió la intensidad de luz de un led en intervalos de tiempo controlados, con lo que se practicó el uso de los ciclos de trabajo. Para probar la variación de frecuencia se implementó el speaker y con un afinador de guitarra se comprobó la precisión de la nota.
    - ADC: Para las pruebas de este conversor se utilizaron potenciómetros que daban una tensión proporcional entre la tensión de alimentación (5V) y masa. Para tener una referencia visual se utilizó la variación de intensidad de un led y más adelante la función meter de la librería de los LEDs.

- Integración del stack USB con las librerías desarrolladas para la placa. Para la validación de esta última fase del proyecto se ha utilizado el videojuego Secret Maryo Chronicles en las tres principales plataformas, MacOsX, Windows7 y Ubuntu 11.04. Se ha comprobado que el Juego, de forma nativa detecta el dispositivo USB y es capaz de reconocer el uso de los botones y la inclinación diferencial como el movimiento de la palanca.

5. Conclusión

Dado el carácter didáctico y divulgativo de este proyecto, las conclusiones que a continuación se recogen, procurarán dar una visión acerca del cómo afrontar proyectos similares.

La programación de microcontroladores, a diferencia de entornos virtuales de más alto nivel, provee de un excelente marco en el que desarrollar conocimientos electrónicos teóricos. Esto es especialmente interesante para un Ingeniero Industrial, que a lo largo de su carrera debe apoyarse en sistemas de terceros para trabajar.

Sin embargo, este tipo de programación puede resultar tediosa a la hora de realizar pruebas, pues no resulta sencillo obtener información de porqué no está funcionando un programa que aparentemente está bien planteado. Para paliar estas dificultades es importante intentar dividir el proyecto en bloques que se puedan probar por separado y desarrollar librerías bien probadas. Es decir, probar la escritura del microcontrolador e ir practicando con cada elemento por separado siguiendo una evolución incremental.

Una vez se ha conseguido dominar la programación de microcontroladores y sus distintos periféricos, los interfaces estándar, como el USB, simplifican enormemente la labor de desarrollo, pues se consigue una gran portabilidad entre sistemas. Más aún, como el caso de este proyecto, si se utilizan fuentes y software libre.

Los entornos de desarrollo o IDEs, simplifican la programación pero en función del sistema operativo y del entorno elegido puede resultar muy laboriosa la tarea de adaptarlo para el desarrollo de software de microcontrolador.

El otro aspecto que se ha podido demostrar en este proyecto es la madurez y alcance del software libre, tanto de los sistemas operativos como de las aplicaciones. Con ligeros matices, resultan tan útiles, potentes y sencillos como los comerciales.

5.1. Presupuesto

Para la realización de este proyecto se ha dispuesto de los siguientes elementos:

- Bobina de estaño: Con bajo, o nulo, contenido en plomo. El plomo confiere a la soldadura mejores propiedades mecánicas pero es altamente tóxico, y por tanto, desaconsejable. Para el proyecto se han gastado unos 25 gramos de una bobina de 200 gramos (diámetro 0,8mm) a un precio de 16€ la bobina
- Soldador 14W: Cualquier soldador para soldadura de baja temperatura (menor de 50W) y con punta fina, es adecuado para esta aplicación. Se debe tener en cuenta, para proyectos de estas características que a mayor potencia, se consigue mayor velocidad en la soldadura, pero también aumenta el riesgo de dañar los componentes y el sustrato donde se suelden los componentes. El soldador tiene un precio de 18€
- Soporte para el soldador: Resulta imprescindible para este tipo de aplicaciones. El soporte evita accidentes con la punta del soldador y la esponja, debidamente humedecida, permite limpiar la punta del soldador y obtener soldaduras limpias. Precio 10€
- Chupón: Si no se dispone de una estación de soldadura, el chupón es una herramienta eficaz para rectificar soldaduras. Tiene un precio de 10€

Y se han adquirido los siguientes componentes, por un total de 53€:

- Zócalo 14 pines 0,5€
- Zócalo 44 pines, 2€
- 8 LEDs rojos, 0,25€ cada, total 2€
- 17 Resistencias de 0,25W: 0,15€ cada, total 2,55€
- 2 Pulsadores normalmente abiertos: 0,5€ cada, 1€ total
- 2 Potenciómetros : 0,5€ cada, 1€ total.
- 1 Potenciómetro multivuelta, 1€
- LM317, 1€
- Buzzer: 1€
- Teensy 2++ con pines: 24$ (+gastos de envío 12$)= 24€
- Acelerómetro LIS302ALB de ST. Descatalogado, uno de similares características 10€
- Array de 40 pines, 2€
- Perfboard (50x40mm), 5€

5.2. Propuestas de ampliación

El proyecto ha sido creado con técnicas de programación bastante limitadas, por lo que la ampliación más importante consistiría en reescribir el código de programa con técnicas de programación más acordes con la técnica actual.

La idea inicial del proyecto fue crear un sustituto libre de un WiiMote, un controlador de la plataforma de videojuegos de Nintendo, Wii. Un proyecto, que al igual que el presente, constase de microcontrolador, LEDs, buzzer y acelerómetro, pero en vez de utilizar USB utilizase la tecnología inalámbrica Bluetooth. Se descartó pues los costes de desarrollo se disparaban por el precio de las placas de entrenamiento de las tecnologías Bluetooth. Sin embargo el desarrollo hubiese sido más interesante, por la versatilidad, y más sencillo, pues por lo general la forma de integrar bluetooth en un proyecto consiste en programar un microcontrolador que se comunica por SPI con un circuito integrado que se encarga de la comunicación bluetooth de manera autónoma.

Durante la investigación de qué microcontrolador utilizar para desarrollar el prototipo, se barajó la tecnología Arduino. Un proyecto de hardware y software libre, que a partir de microcontroladores ATMega crea un entorno de programación extremadamente simplificado con gran cantidad de librerías y aplicaciones. Una ampliación a sugerir sería el desarrollo y documentación de este proyecto con esta filosofía.

En la actualidad cada vez es más frecuente el uso de pantallas táctiles, y una tecnología especialmente elegante y muy accesible para proyectos universitarios es la óptica a base de LEDs infrarrojos. La teoría apunta a que un led puede funcionar también como receptor de luz y que los paneles LCD de los monitores no filtran la luz infrarroja, por lo que si se pudiese acondicionar una matriz de LEDs infrarrojos a modo de backlight se tendría una pantalla táctil muy robusta.

Página dejada en blanco intencionadamente

6. Anexos

6.1. Microcontrolador

6.1.1. Historia

En 1971 la compañía "Intel" lanzó al mercado el 4004, el primer microcontrolador de la historia. Un circuito lógico secuencial síncrono, que a diferencia de sus contemporáneos, presentaba una combinación adecuada de los códigos de entrada, ejecución secuencial, poder saltar hacia atrás o adelante en esta secuencia mediante decisiones programadas y permitía realizar una gran diversidad de ejecuciones complejas en simples códigos.

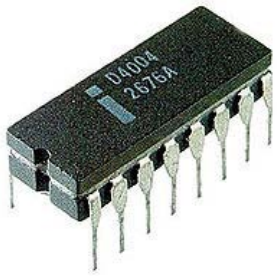

Figura 93. Fotografía de un microcontrolador 4004

Aunque no fuese más que la evolución e integración de tecnologías existentes, hasta el punto de que "Busicom", la compañía china que lo encargó no mostró gran interés, supuso una revolución cientifico-técnica porque permitió desarrollar aplicaciones de una manera más eficaz, rápida y barata. La evolución de estos sistemas y la necesidad de especialización de sus elementos dieron lugar a la aparición de los microprocesadores y al resto de sistemas.

6.1.2. Arquitecturas

En la actualidad existen dos arquitecturas: Von Neumann y Hardvard. Ambas se diferencian en la forma de la conexión de la memoria al procesador y en los buses que necesita.

a. **Von Neumann** sería la arquitectura del concepto de microprocesador arriba explicado y es en el que se basan los ordenadores personales. Es decir una arquitectura segregada.

   Según esta arquitectura, cuando se carga un programa en memoria, a éste se le asigna un espacio de direcciones de la memoria que se divide en segmentos, de los cuales típicamente tenderemos los siguientes: código (programa), datos y pila. Es decir utiliza la memoria como un todo y no distingue entre interna y externa.

b. La otra variante es la arquitectura **Harvard,** y por excelencia la utilizada en superordenadores, en los microcontroladores, y sistemas integrados en general. En este caso, además de la memoria, el procesador tiene los buses segregados, de modo que cada tipo de memoria tiene un bus de datos, uno de direcciones y uno de control.

La ventaja fundamental de esta arquitectura es que permite adecuar el tamaño de los buses a las características de cada tipo de memoria; además, el procesador puede acceder a cada una de ellas de forma simultánea, lo que se traduce en un aumento significativo de la velocidad de procesamiento, típicamente los sistemas con esta arquitectura pueden ser dos veces más rápidos que sistemas similares con arquitectura Von Neumann.

La desventaja está en que consume muchas líneas de E/S del procesador; por lo que en sistemas donde el procesador está ubicado en su propio encapsulado, solo se utiliza en superordenadores. Sin embargo, en los microcontroladores y otros sistemas integrados, donde usualmente la memoria de datos y programas comparten el mismo encapsulado que el procesador, este inconveniente deja de ser un problema serio y es por ello que encontramos **la arquitectura Harvard en la mayoría de los microcontroladores.**

6.1.3. Estructura de un microcontrolador

La estructura de los microcontroladores puede variar mucho dependiendo del fabricante y del modelo del integrado. Existen formas, quizás más coherentes, de organizar la estructura, no obstante con la estructura que se detalla a continuación pretende ser una visión generalista desde una perspectiva funcional, orientado a los microcontroladores de arquitectura Harvard.

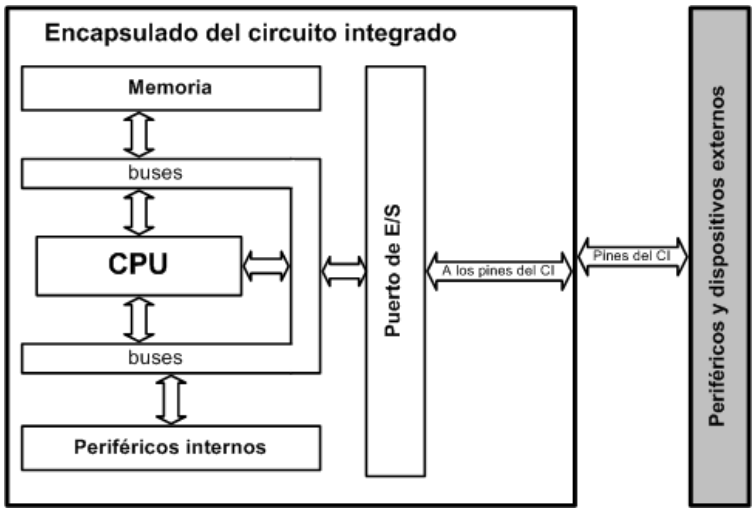

Figura 94. Esquema de la estructura de un microcontrolador

En los siguientes apartados se desarrollarán los distintos elementos emplazados según esta estructura.

6.1.4. CPU: Unidad Central de Proceso

a. Unidad de control

Es el el intérprete y árbitro de las instrucciones. Son un elemento muy complejo por lo que tiende a dividirse en unidades más pequeñas según su función y estas son: la unidad de decodificación, unidad de ejecución, controladores de memoria cache, controladores de buses, controladores de interrupción, pipelines, entre otros elementos.

Existen tres tipos de CPU en cuanto a la forma de "procesar" las instrucciones:

- **CISC**: Un gran número de procesadores usados en los microcontroladores están basados en la filosofía CISC (Computadores de Juego de Instrucciones Complejo). Disponen de más de 80 instrucciones de máquina en su repertorio, algunas de las cuales son muy sofisticadas y potentes, pero requieren muchos ciclos para su ejecución. Una ventaja de los procesadores CISC es que ofrecen al programador instrucciones complejas que actúan como macros.
- **RISC:** Tanto la industria de los ordenadores comerciales como la de los microcontroladores, están decantándose hacia la filosofía RISC (Computadores de Juego de Instrucciones Reducido). En estos procesadores el repertorio de instrucciones de máquina es muy reducido y las instrucciones son simples y, generalmente, se ejecutan en un ciclo. La sencillez y rapidez de las instrucciones permiten optimizar el hardware y el software del procesador.
- **SISC:** En los microcontroladores destinados a aplicaciones muy concretas, el juego de instrucciones, además de ser reducido, es "específico"; o sea, las instrucciones se adaptan a las necesidades de la aplicación prevista. Esta filosofía se ha bautizado con el nombre de SISC (Computadores de Juego de Instrucciones Específico).

b. Unidad aritmético-lógica

Aquí es donde se realizan las sumas, restas y operaciones lógicas del álgebra de Boole. Es una parte tan importante, por su volumen de uso, que en los ordenadores personales, en los microprocesadores, se tiende a separar de la CPU y se le denomina cooprocesador matemático, "FPU".

c. Interrupciones.

Pese a no ser un elemento propiamente dicho, la gestión de interrupciones, por parte de la unidad de control, implica la atención de eventos sucedidos en el microcontrolador, y dada su naturaleza excepcional se ha creído interesante separar del resto de conceptos de la CPU.

Estos eventos pueden ser internos, producidos por un temporizador o contador, o bien externos, por ejemplo por la excitación de un PIN del microcontrolador. Según la interrupción activada, el programa que se ejecuta de manera secuencial, saltará a una posición específica de la memoria, a su vector de interrupción, explicado más adelante.

Los procesos de atención a interrupciones tienen la ventaja de que se implementan por hardware ubicado en el procesador, así que es un método rápido de hacer que el procesador se dedique a ejecutar un programa especial. Suponen una manera eficaz de ahorrar cálculos al procesador, pues éste no debe estar comprobando periódicamente el evento para actuar en consecuencia. Si no que el procesador es avisado y puesto en situación cuando se activa tal evento.

En términos generales, un proceso de interrupción y su atención por parte del procesador, tiene la siguiente secuencia de acciones:

1. En el mundo real se produce el evento para el cual se requiere que el procesador ejecute un programa especial, este proceso tiene la característica de que no puede esperar mucho tiempo antes de ser atendido o no se sabe en que momento debe ser atendido.

2. El circuito encargado de detectar la ocurrencia del evento se activa, y como consecuencia, activa la entrada de interrupción del procesador.

3. La unidad de control detecta que se ha producido una interrupción y “levanta” una bandera para registrar esta situación; de esta forma si las condiciones que provocaron el evento desaparecen y el circuito encargado de detectarlo desactiva la entrada de interrupción del procesador, ésta se producirá de cualquier modo, porque ha sido registrada.

4. La unidad de ejecución termina con la instrucción en curso y justo antes de comenzar a ejecutar la siguiente comprueba que se ha registrado una interrupción

5. Se desencadena un proceso que permite guardar el estado actual del programa en ejecución y saltar a una dirección especial de memoria de programas, donde está la primera instrucción de la subrutina de atención a interrupción.

6. Se ejecuta el código de atención a interrupción, esta es la parte “consciente” de todo el proceso porque es donde se realizan las acciones propias de la atención a la interrupción y el programador juega su papel.

7. Cuando en la subrutina de atención a interrupción se ejecuta la instrucción de retorno, se desencadena el proceso de restauración del procesador al estado en que estaba antes de la atención a la interrupción.

6.1.5. Entradas y salidas de propósito general

También conocidos como puertos de entrada/salida ("E/S"), generalmente agrupadas en puertos de 8 bits de longitud, permiten leer datos del exterior o escribir en ellos desde el interior del microcontrolador. Es decir, 8 pines que representan un bit (1 ó 0) de información.

Algunos puertos de E/S tienen características especiales que le permiten manejar salidas con determinados requerimientos de corriente, o incorporan mecanismos especiales de interrupción para el procesador.

Típicamente cualquier pin de E/S puede ser considerada E/S de propósito general, pero como los microcontroladores no pueden tener infinitos pines, las E/S de propósito general comparten los pines con otros periféricos. Para usar un pin con cualquiera de las características a él asignadas debemos configurarlo mediante los registros destinados a ello.

6.1.6. Buses

Es el medio de comunicación que utilizan los diferentes componentes del procesador para intercambiar información entre sí. Existen tres tipos de buses, de dirección, de datos y de control. Su implementación es meramente eléctrica y es transparente para el programador por lo que no se detallará más.

6.2. Programación de Microcontroladores

NOTA. Las generalidades de programación que se exponen más abajo están referidas a los microcontroladores AVR, aunque salvo raras excepciones, se puede extrapolar al resto de familias.

6.2.1. Introducción

Como se ha explicado antes, los microcontroladores procesan el programa como si fuese una pila. Es decir, en principio, desde la primera hasta la última línea en orden. Estas líneas empiezan con una orden ("opcode") y seguido, si procede, de los operandos, por tanto una línea de código que el microprocesador podría reconocer sería:

10110000 01100001

El procesador, llegado a este punto, copiaría el valor hexadecimal 61, al registro "AL". Pues el primer 1011 equivale a la operación "mover", el siguiente 0000 sería el identificador del registro "AL", y 01100001 se corresponde con el número 97 (61 en hexadecimal). Tras procesar esta orden saltaría a la siguiente línea y la ejecutaría de la misma forma.

6.2.2. Lenguaje Ensamblador

Para simplificar la labor del programador, existe una aplicación capaz de traducir reglas mnemotécnicas en código máquina. Por tanto el programador, en lugar del código binaro de arriba, debe escribir:

MOV AL, 61h

Y la aplicación lo traducirá. Esta aplicación se conoce como "Ensamblador" y el lenguaje de reglas mnemotécnicas como "Lenguaje ensamblador".

Fue creado por Dennis Ritchie en el año 1969, y se considera un lenguaje de bajo nivel pues está más cerca del entendimiento de un procesador que el de un ser humano. Es muy dependiente del procesador que se intenta programar, por lo que la portabilidad a otro procesador puede ser complicada.

En la actualidad está en desuso a nivel masivo, pues sigue siendo muy lento de programar para grandes proyectos, sin embargo es muy útil cuando se necesitan aplicaciones de alto rendimiento pues se tiene un control absoluto sobre el procesador.

6.2.3. ANSI C

Historia

C es un lenguaje de programación creado en 1972 por Dennis M. Ritchie en los Laboratorios Bell. C es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones.

En 1983, el Instituto Nacional Estadounidense de Estándares organizó un comité, X3j11, para establecer una especificación estándar de C. Tras un proceso largo y arduo, se completó el estándar en 1989 y se ratificó como el "Lenguaje de Programación C" ANSI X3.159-1989. Esta versión del lenguaje se conoce a menudo como ANSI C, o a veces como C89 (para distinguirla de C99).

Características

Se trata de un lenguaje débilmente tipificado de medio nivel pero con muchas características de bajo nivel. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos.

Lenguaje

Ahora las órdenes implican varias acciones de la CPU. Y se hace más comprensible para el ser humano. Por ejemplo, la orden “MOV AL, 61h”, podría ser equivalente a por ejemplo:

AL = 61;

Y se introducen nuevos conceptos como funciones con argumentos, de tal forma que si entendemos la palabra “int” (numero entero) como la especificación del tipo de dato al que precede, es trivial entender que si se invoca la función sumar con dos operadores, dará un resultado. Se ahorra de esta forma mucho código y se simplifica la lectura.

```
int sumar (int sumando1, int sumando2)    //definición de la función
{
        return (sumando1 + sumando2);    // procesamiento de los operadores
}
...
b = sumar (1, 2) ;                        //utilización
```

Se añade el concepto de las condiciones y bucles, queson funciones ampliamente utilizadas por su utilidad. Como la condición “If (a == b) {a = a +1}”, que le suma 1 a la variable “a” si se cumple la condición ”a = b”.

Otra característica muy interesante es la posibilidad de pasar valores por argumento o por valor a una función, ya que cada vez que invocamos una función con operandos, esta los duplica para trabajar con ellos sin modificarlos si es por valor, no así cuando es por argumentos. Esto se desprende de una de las grandes potencias de este lenguaje, los punteros.

Se podría continuar desarrollando esta explicación indefinídamente, pero no es el motivo de estudio de este proyecto. Existen excelentes tutoriales de programación en C gratuitos en internet, uno de ellos es c.conclase [10]

Utilización y creación de librerías

Los lenguajes de programación, permiten, para facilidad del programador, organizar los archivos de texto creados. De tal forma que si se escriben una serie de funciones para controlar, por ejemplo un LCD, se pueden organizar en un mismo archivo de texto para reutilizarlo con otro programa.

De hecho, permite la creación de dos tipos de fichero para las mismas funciones. El fichero de cabeceras ".h" que contiene los prototipos de las funciones (sólo la definición según el anterior ejemplo) y las definiciones. Y por otro lado el archivo ".c" que contiene las funciones completas. Para que el archivo a compilar llame a estos otros archivos se utiliza la orden:

#include

En este sentido, por ejemplo, el archivo "lcd.h" sirve para la presentación con comentarios de descripción literaria de las funciones y para la configuración a través de los "defines". Además este archivo contendría la llamada al archivo "lcd.c" que contendría las funciones desarrolladas y no necesariamente explicadas

6.2.4. C++

Introducción

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido.

Posteriormente se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje de programación multiparadigma.

El nombre C++ fue propuesto por Rick Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "C con clases". En C++, la expresión "C++" significa "incremento de C" y se refiere a que C++ es una extensión de C.

POO

La programación orientada a objetos o “POO”, es un paradigma de programación que implica la creación de objetos a partir de clases, a diferencia de las anteriores que serían “estructuradas”. Se podría explicar vagamente el concepto de clase como una función que en lugar de ofrecer resultados, crea objetos. Estos objetos son una “especie de variable”, que tiene atributos propios y funciones heredadas de la clase (métodos).

Si por ejemplo existiese una clase “coche”, un objeto podría ser “mi_coche” y a este objeto se le podría ordenar a través del método “acelerar” para que variase su atributo “posición” y “nivel de gasolina”.

Esta forma de programación permite el encapsulamiento, para la reutilización de código y la abstracción, ya que un segundo programador que no conozca cómo funciona internamente una clase, podría manejarla sin problemas. De hecho es de la forma con la que actualmente se desarrollan las aplicaciones de relativa complejidad.

Página dejada en blanco intencionadamente

7. Agradecimientos

Por supuesto, a mi tutor de este proyecto, Luis Entrena, sin su flexibilidad y paciencia este proyecto hubiese sido inviable.

A mis compañeros de carrera, en especial Luis García y Jose Antonio Jiménez, mi inspiración.

A mis amigos, todos, pero en especial, Juan Antonio Espinosa de los Monteros y Carlos Maseda, mi apoyo.

A mi jefe, José Miguel Huertas, por devolverme la ambición del Ingeniero.

A mi amiga, Rosa Mora, por demostrarme que creatividad y profesionalidad no están reñidas.

A mi consultor tecnológico personal, Dean Camera, por su inestimable ayuda y consejo.

A los contribuidores de licencias libres anónimos, este proyecto es mi tributo a ellos.

A mi familia, en especial a mis padres, por todo lo anterior y más.

Página dejada en blanco intencionadamente

8. Bibliografía

8.1. Libros

- [1] "USB Complete Third Edition: Everyting You Need to Develop Custom USB Peripherals" de Jan Axelson
- [2] "Embedded C Programming and the Atmel AVR", Richard Barnett, Larry O´Cull, Sarah Cox.
- [3] "Resolución de problemas con C++, El objetivo de la programación". Segunda edición de Walter Savitch
- [4] "C++ Curso de programación" Segunda Edición . Fco. Javier Ceballos, editorial Ra-Ma
- [5] W. Stallings. "Sistemas Operativos". 5ª Edición, Ed. Pearson

8.2. Datasheets:

- [6] Acelerómetro LIS302ALB: http://pdf1.alldatasheet.com/datasheet-pdf/view/171330/STMICROELECTRONICS/LIS302ALB.html
- [7] Regulador de tensión LM317 http://www.national.com/ds/LM/LM117.pdf
- [8] Microcontrolador AT90USB1286 http://www.atmel.com/dyn/resources/prod_documents/doc7593.pdf

8.3. Páginas de Internet consultadas:

- [9] Comunidad de programadores de microcontroladores Atmel http://www.avrfreaks.net/
- [10] Recurso web con cursos y lecciones de programación en C y C++ http://c.conclase.net/
- [11] Web del producto con recursos para la programación del microcontrolador USB utilizado http://pjrc.com/
- [12] Web del stack USB utilizado. http://www.fourwalledcubicle.com/LUFA.php

8.4. Repositorio del proyecto

- http://code.google.com/p/pfcuc3m/

Todas las imágenes, salvo que se especifique lo contrario en la propia descripción, están disponibles en internet con licencias libres tales como las Creative Commons.